Accessible software frameworks for reproducible image analysis of host-pathogen interactions

DISSERTATION

zur Erlangung des akademischen Grades eines

„doctor rerum naturalium“ (Dr. rer. nat.)

vorgelegt dem Rat der Fakultät für Biowissenschaften der Friedrich-Schiller-Universität Jena

von Ruman Gerst

geboren am 19.05.1993 in Alsfeld

Gutachter

1. Prof. Dr. Marc Thilo Figge (Leibniz-Institut für Naturstoff-Forschung und Infektionsbiologie, Jena, Deutschland)
2. Prof. Dr. Peter Zipfel (Leibniz-Institut für Naturstoff-Forschung und Infektionsbiologie, Jena, Deutschland)
3. Prof. Dr. Sabine Fischer (Center for Computational and Theoretical Biology, Würzburg, Deutschland)

zusätzliche Mitglieder der Kommission zur Verteidigung am 13.12.2022

1. Prof. Dr. Günter Theißen (Matthias-Schleiden-Institut, Friedrich-Schiller-Universität Jena, Jena, Deutschland)
2. Prof. Dr. Stefan Schuster (Matthias-Schleiden-Institut, Friedrich-Schiller-Universität Jena, Jena, Deutschland)
3. Prof. Dr. Steve Hoffmann (Leibniz-Institut für Alternsforschung, Jena, Deutschland)

Contents

Summary

To gain a full understanding of life-threatening diseases, it is essential to fully characterize the interactions between host cells and pathogenic microorganisms. The ongoing advancements in imaging techniques and computer hardware allow the implementation of image-based systems biology (IbSB) that is based around the extraction of precise measurements from visualizations of cells, tissues, or whole organs via advanced software tools. To meet the current standards of digital research data management, the software must implement the principles of *Findability, Accessibility, Interoperability, and Reusability* (FAIR) and contribute towards their propagation in the scientific community. This is especially important for interdisciplinary teams of experimentalists and computer scientists which benefit from computer programs that bridge communication gaps, and facilitate the adoption of new technologies. In this thesis, software frameworks were introduced that contribute towards the propagation of FAIR principles in the field of IbSB by the development of standardized, reproducible, high-performance, and accessible software for the quantification of interactions in biological systems.

The subject of the first study was multispectral optoacoustic tomography (MSOT), which is a non-invasive imaging method capable of measuring pharmacokinetics *in vivo*. We aimed to confirm the viability of MSOT for the differentiation between healthy and septic mice by tracking the liver function via the dynamics of indocyanine green (ICG). The standard approach for the quantification of MSOT data relies on manually defined regions of interest (ROIs), making the analysis time consuming and hard to reproduce. Additionally, the study showed that the manual approach was sensitive to small changes in the ROIs and was generally unsuccessful in finding significant differences between the extracted measurements. We thus developed an algorithm termed *MSOT cluster analysis toolkit* (*Mcat*) that allows the fully automated quantification of MSOT images by a combination of two machine-learning methods. The automated approach successfully differentiated septic from healthy mice based on ICG signal kinetics. The algorithm was distributed as plugin for the *ImageJ* platform, thus making the automated analysis of MSOT data accessible to non-programmers.

The second publication addressed a general disadvantage of the *ImageJ* platform: while one of its hallmarks is the user-friendly graphical user interface (GUI), reproducible, automated, and FAIR-compliant image processing pipelines can only be created in a script language. Consequently, this reliance on programming hinders the discourse within interdisciplinary teams and their adoption of FAIR guidelines. To bridge the gap between computer- and life-scientists, we encapsulated

the functionality of *ImageJ* into a visual programming language termed *Java image processing pipeline* (*JIPipe*). It combines the reproducibility of programming languages with the accessibility of flow charts, and thus opens the design of advanced quantitative image analysis pipelines to non-programmers. Beyond the capabilities of *ImageJ*, *JIPipe* also introduces standardized formats for the storage of data and algorithm parameters that further simplify the implementation of the FAIR principles.

JIPipe was utilized in the third study that aimed to characterize the defense mechanism of the soil-dwelling fungus *Mortierella verticillata* against the fungivorous nematode *Aphelenchus avenae*. We discovered a new species of endosymbiotic bacteria that produces cytotoxic macrolactones and showed through viability assays that the newly identified secondary metabolite inhibits both *Aphelenchus avenae* and *Caenorhabditis elegans*. These finding were realized due to the application of a fully automated image analysis pipeline based on *JIPipe*, which could identify and track the nematodes with the purpose of quantifying their mobility.

The fourth publication addressed the challenges in developing high-performance image analysis pipelines. Light sheet fluorescence microscopy and other modern imaging techniques produce big volumes of data that are only processable by purpose-built and highly optimized algorithms. While the machine-oriented programming language *C++* already enables the development of such tools, these are usually not easily accessible to non-programmers. To facilitate the standardization, FAIR-compliance, and ease of access to high-performance software, we developed a framework termed *modular image stack analysis for C++* (*MISA++*) that simplifies the design of optimized image analysis pipelines via standardized components. These allowed the creation of a user-friendly GUI around any software developed with the *MISA++* framework.

In summary, this thesis showcases how software frameworks can contribute towards the characterization of host-pathogen interactions by simplifying the design and application of quantitative image analysis pipelines according to the FAIR principles. These developments are the ideal starting points for future collaborations with life scientists and medical doctors that will lead to the development of improved experiments, imaging methods, algorithms, and computer models in the spirit of the image-based systems biology approach.

Zusammenfassung

Um die Mechanismen hinter lebensgefährlichen Krankheiten zu verstehen, müssen die zugrundeliegenden Interaktionen zwischen den Wirtszellen und krankheitserregenden Mikroorganismen bekannt sein. Die kontinuierlichen Verbesserungen in bildgebenden Verfahren und Computertechnologien ermöglichen die Anwendung von Methoden aus der bildbasierten Systembiologie, welche moderne Computeralgorithmen benutzt um das Verhalten von Zellen, Geweben oder ganzen Organen präzise zu messen. Um den Standards des digitalen Managements von Forschungsdaten zu genügen, müssen Algorithmen den *FAIR*-Prinzipien (*Findability, Accessibility, Interoperability, and Reusability*) entsprechen und zur Verbreitung ebenjener in der wissenschaftlichen Gemeinschaft beitragen. Dies ist insbesondere wichtig für interdisziplinäre Teams bestehend aus Experimentatoren und Informatikern, in denen Computerprogramme zur Verbesserung der Kommunikation und schnellerer Adaption von neuen Technologien beitragen können. In dieser Arbeit wurden daher Software-Frameworks entwickelt, welche dazu beitragen die *FAIR*-Prinzipien durch die Entwicklung von standardisierten, reproduzierbaren, hochperformanten, und leicht zugänglichen Softwarepaketen zur Quantifizierung von Interaktionen in biologischen System zu verbreiten.

Der Gegenstand der ersten Studie ist das nicht-invasive Bildgebungsverfahren der multispektralen optoakustischen Tomografie (MSOT), welches in der Lage ist Pharmakokinetiken *in vivo* zu erfassen. Da die Dynamik von Indocyaningrün in Verbindung mit der Leberfunktion steht, sollten daher gesunde von septische Mäusen anhand von MSOT unterschieden werden. Das Standardverfahren zur Analyse von den produzierten Bilddaten basiert auf einer von einem Experten manuell definierten Region, was den Prozess verlangsamt und schwer reproduzierbar macht. Die Studie konnte zudem zeigen, dass das klassische Verfahren empfindlich gegenüber kleinen Änderungen in den Regionen ist und teilweise nicht in der Lage war signifikante Unterschiede zwischen Messungen zu erkennen. Daher wurde der Algorithmus *MSOT cluster analysis toolkit* (*Mcat*) entwickelt, der durch die Kombination von zwei Methoden des maschinellen Lernens eine vollautomatische Quantifizierung von MSOT-Daten erzielt. Diese Methode war in der Lage septische von gesunden Mäusen korrekt zu unterscheiden. Eine für Nichtprogrammierer zugängliche Implementierung wurde in Form einer *ImageJ*-Erweiterung bereitgestellt.

Die zweite Publikation befasst sich mit einem generellen Problem der *ImageJ*-Plattform: während einerseits durch eine grafische Benutzeroberfläche die Benutzung erleichtert wird, so ist andererseits der Entwurf von reproduzierbaren, au-

tomatischen und *FAIR*-kompatiblen Bildverarbeitungswerkzeugen nur durch Programmierung möglich. Da Skriptsprachen nicht jedem zugänglich sind, wird sowohl die Kommunikation, als auch die Umsetzung der *FAIR*-Prinzipien in interdisziplinären Teams erschwert. Aus diesem Grund wurde die visuelle Programmiersprache *Java image processing pipeline* (*JIPipe*) entwickelt, welche es Nutzern erlaubt Funktionen von *ImageJ* mithilfe eines Flussdiagramms anzuordnen. Dadurch wird auch Nichtprogrammierern der Entwurf von quantitativen und *FAIR*-kompatiblen Bildverarbeitungspipelines ermöglicht.

JIPipe wurde in der dritten Studie angewendet, deren Gegenstand die Charakterisierung eines Verteidigungsmechanismus des Bodenpilzes *Mortierella verticillata* gegen den pilzfressenden Fadenwurm *Aphelenchus avenae* war. Es wurde eine neue Spezies von endosymbiontischen Bakterien entdeckt, welche zytotoxische Makrolactone produzieren. Deren Giftigkeit gegenüber *Aphelenchus avenae* und *Caenorhabditis elegans* wurde mithilfe von Experimenten und darauffolgender Bildanalyse von Mikroskopiebildern festgestellt. Die Quantifizierung der Daten wurde durch eine vollautomatische Pipeline basierend auf *JIPipe* durchgeführt, die in der Lage war individuelle Fadenwürmer zu verfolgen und deren Bewegungsverhalten zu quantifizieren.

Die vierte Publikation adressiert die Probleme in der Entwicklung von hochperformanten Bildanalyseprogrammen. Moderne Bildgebungsverfahren wie Lichtscheibenmikroskopie erzeugen große Datenvolumen und können nur von speziell optimierten Algorithmen hinreichend schnell quantifiziert werden. Die maschinennahe Programmiersprache *C++* erlaubt bereits die Entwicklung von hochperformanten Werkzeugen, welche aber oft nur von Programmierern bedient werden können. Daher wurde das Software-Framework *modular image stack analysis for C++* (*MISA++*) entwickelt, das standardisierte Komponenten zur vereinfachten Entwicklung von standardisierten, hochperformanten und *FAIR*-kompatiblen Bildanalyseprogrammen bereitstellt. Durch die Standardisierung war es möglich, eine einheitliche Benutzeroberfläche für jedes auf *MISA++* basierende Programm zur Verfügung zu stellen.

Zusammenfassend zeigt diese Arbeit wie Software-Frameworks zu der Charakterisierung von Interaktionen zwischen Wirtszellen und Pathogenen beitragen können, indem der Entwurf und die Anwendung von quantitativen und *FAIR*-kompatiblen Bildanalyseprogrammen vereinfacht werden. Diese Verbesserungen erleichtern zukünftige Kollaborationen mit Lebenswissenschaftlern und Medizinern, was nach dem Prinzip der bildbasierten Systembiologie zur Entwicklung von neuen Experimenten, Bildgebungsverfahren, Algorithmen, und Computermodellen führen wird.

Glossary

e.g. *exempli gratia* (for example)

i.e. *id est* (that is)

Acronyms

1D one-dimensional

2D two-dimensional

3D three-dimensional

4D four-dimensional

ABM agent-based model

API application programming interface

CLAHE contrast limited adaptive histogram equalization

CLI command line interface

CNN convolutional neural network

CPU central processing unit

DNA deoxyribonucleic acid

DNN deep neural network

FAIR *Findability, Accessibility, Interoperability, and Reusability*

GFP green fluorescent protein

GPU graphics processing unit

GUI graphical user interface

HTTP hypertext transfer protocol

IbSB image-based systems biology

IC_{50} inhibitory concentration at 50%

ICG indocyanine green

JIPipe *Java image processing pipeline*

JNI *Java Native Interface*

JSON *Javascript* object notation

LED light-emitting diode

LSFM light sheet fluorescence microscopy

Mcat *MSOT cluster analysis toolkit*

MISA++ *modular image stack analysis for C++*

MSOT multispectral optoacoustic tomography

PSF point spread function

RAM random access memory

RDM research data management

ReLU rectified linear unit

REST *Representational State Transfer*

RNA ribonucleic acid

ROI region of interest

ROS reactive oxygen species

TL transmitted light

URL uniform resource locator

VM virtual machine

VPL visual programming language

vRAM video random access memory

Chapter 1

Introduction

The invention of the microscope around 1600 [118, Appendix 6, Page 787] provided humanity with the means to resolve objects beyond the capabilities of their eyes or simple magnifying glasses. This lead to the first description of microorganisms by Robert Hooke [125, Chapter 1.6, Page 13], who among other things produced the first depictions of mold fruiting bodies and compartmentalized structures termed "cells". While the first bacteria were already discovered 1676 by Antoni van Leeuwenhoek, the role of microorganisms as vectors of disease was not yet established. An important factor that contributed towards the further characterization of diseases was the continued improvements of microscopes that include the correction against spherical aberrations by Joseph Jackson Lister, the establishment of the diffraction theory by Ernst Carl Abbe, and the development of a uniform illumination technique by August Köhler [118, Appendix 6, Page 787]. The improvements still continue to this day and for example, lead to methods for resolving objects below the diffraction limit [58]. While the microscope technology improved, Louis Pasteur showed that microorganisms do not form spontaneously [125, Chapter 1.7, Page 15] and must be introduced into an environment, thus establishing the concept of sterilization. The proof that specific microorganisms are the causative agents of diseases was provided by Robert Koch, who successfully isolated *Bacillus anthracis*, cultivated the bacterium *in vitro*, and showed by the application of an animal model that infections can be transferred between hosts [125, Chapter 1.8, Page 16]. The newly developed methods for cultivating microorganisms were then successfully applied to identify *Mycobacterium tuberculosis* and *Vibrio cholerae* as the causes of tuberculosis and cholera respectively [125, Chapter 1.8, Page 17]. The new field of microbiology and its methods continued to advance, recognizing that microorganisms are part of complex cross-species interaction networks [125, Chapter 1.9, Page 20].

It is not trivial to characterize the processes and rules behind these interactions, for example, how hosts can defend against pathogenic microorganisms. The reason behind this is that to gain information from an interaction, one cannot merely observe the behavior of a singular cell. Instead, the characteristics only emerge if multiple agents are put into the same environment. This principle of where the "whole is not the same as the sum of its parts" (Aristotle) [7] is the fundament of modern systems biology [72, 80] that aims to characterize a whole system of mul-

tiple players and their interactions on a holistic level [70], i.e., how they contribute towards the function of the system [21]. Bruggeman and Westerhoff further classify systems biology into "top-down" approaches which involve the integration of large data volumes, and methods based on the "bottom-up" principle that predicts emerging interactions from precise molecule-level models. An implementation of the top-down approach is enabled by the ongoing improvements in the characterization of whole genomes [98], proteomes [4], and metabolomes [68] that allow the description of system components and their behavior by the analysis and integration of such "omics" data. For example, Kitano, Levesque and Benfey focus on the study of genetic networks, with the purpose of understanding the interactions between proteins, deoxyribonucleic acid (DNA), and ribonucleic acid (RNA). A notable aspect of systems biology is the close collaboration between computer scientists and experimentalists: the required data can only be obtained from wet lab experiments that produce large volumes of data, thus requiring the utilization of computer tools designed by computer scientists. The result is an iterative cycle [71] which begins with the experiments in the wet lab that are analyzed to produce a high abundance of data. This is followed by the creation of computer models for the generation of new hypotheses, thus leading to the development of new experiments.

The systems biology approach by Kitano, Levesque and Benfey focus on interaction between molecules within a single cell. Over time, great advancements were made in the investigation of interspecies interactions [124] based on "omics" data, including approaches that infer molecular interactions from the transcriptome, proteome, or metabolome. These methods allowed the characterization of, for example, the cellular responses in bacteria that inhabit oral cavities [124], or the relationship between gut bacteria and the fungus *Candida albicans* [94]. A disadvantage of an "omics"-based approach is that higher-level functions need to be reconstructed from molecular reactions between metabolites, DNA, and proteins, which is comparable to the "bottom-up" approach established by Bruggeman and Westerhoff. This makes it difficult to model interactions that have a spatial component and involve a high number of different yet unknown processes.

An alternative is to infer the functionality by the quantification of microscopy or tomography images that capture the interactions of multiple single cells, structures within whole organs, or even pharmacokinetics within whole animals. This concept of image-based systems biology (IbSB) [93] thus implements a "top-down" alternative to the extraction and analysis of "omics" data that incorporates the spatial component of characterizing and modelling cell interactions. Similar to the systems biology approach presented by Kitano, IbSB is implemented as iterative cycle beginning with a laboratory experiment that here produces data in form of images (see Figure 1.1). These can be sourced from a wide range of imaging methods, including wide-field, confocal, and super-resolution microscopy. Images are analyzed with computer programs that apply steps for enhancing the contrast, algorithms for segmenting and tracking the objects of interest, and the extraction of quantitative characterizations of the interactions. As measurements cannot always give a full insight into the underlying processes, computer models are utilized to infer parameters unavailable to image-based measurements. In combination with

the capability of computer models of simulating a system without involving laboratory experiments, new hypotheses can be generated that form the basis of future experiments, thus starting a new cycle.

We have to note that the cyclic concept of IbSB is idealized and not always applicable to all projects. For example, Pollmächer and Figge generated a fungal infection model of a human alveolus based on existing literature [110, 109], leaving out the acquisition and analysis of image data. Due to the available literature, the model could be iteratively improved to simulate processes in a spatial environment that are highly difficult to capture *in vivo*. The model was expanded to encompass the characteristics of murine lungs, revealing differences in the effectiveness of infection clearing [16]. Another study based on a revised model showed the importance of Pores of Kohn in the clearance of infections [17].

A similar implementation of an "incomplete cycle" can be observed in the application of laboratory experiments with subsequent image acquisition and analysis. For example, Kraibooj et al. developed the initial implementation of a software tool to quantify the uptake of *Aspergillus fumigatus* spores by murine macrophages [74]. The analysis relied on the staining of macrophages, which is a time-consuming process. Consequently, new laboratory experiments were performed and resulted in an enhanced image analysis software that is capable of detecting macrophages without requiring a contrast-enhancing agent [32]. These improvements allowed the implementation of a third study that proved that staining impacts the quantitative results [33], thus highlighting the need for advanced image analysis algorithms that are capable of extracting quantitative results from low-contrast images.

Additional challenges in the analysis of images are introduced by the characteristics of newly developed imaging systems that include the handling of data with a size on a terabyte-level [111] according to the principles of *Findability, Accessibility, Interoperability, and Reusability* (FAIR), the complexity in reconstructing images obtained from single molecule localization microscopy [117] or multispectral optoacoustic tomography (MSOT) [133], and the difficulties in establishing a dialogue between experimentalists and bioimage analysis software developers [90].

This thesis aims to contribute towards the resolution of these challenges by the implementation of standardized, reproducible, high-performance, and accessible software for the quantification of interactions in biological systems.

First, we will introduce the term "host-pathogen interaction", and explain its basic mechanisms with the focus on the human immune system (see section 1.1). In section 1.2, we will give an overview of image acquisition techniques, including methods for microscopy and tomography. This is followed by an introduction into the basic operations for image processing and available software implementations (see section 1.3). Finally, we will briefly explain technologies for the development of web applications (see section 1.4).

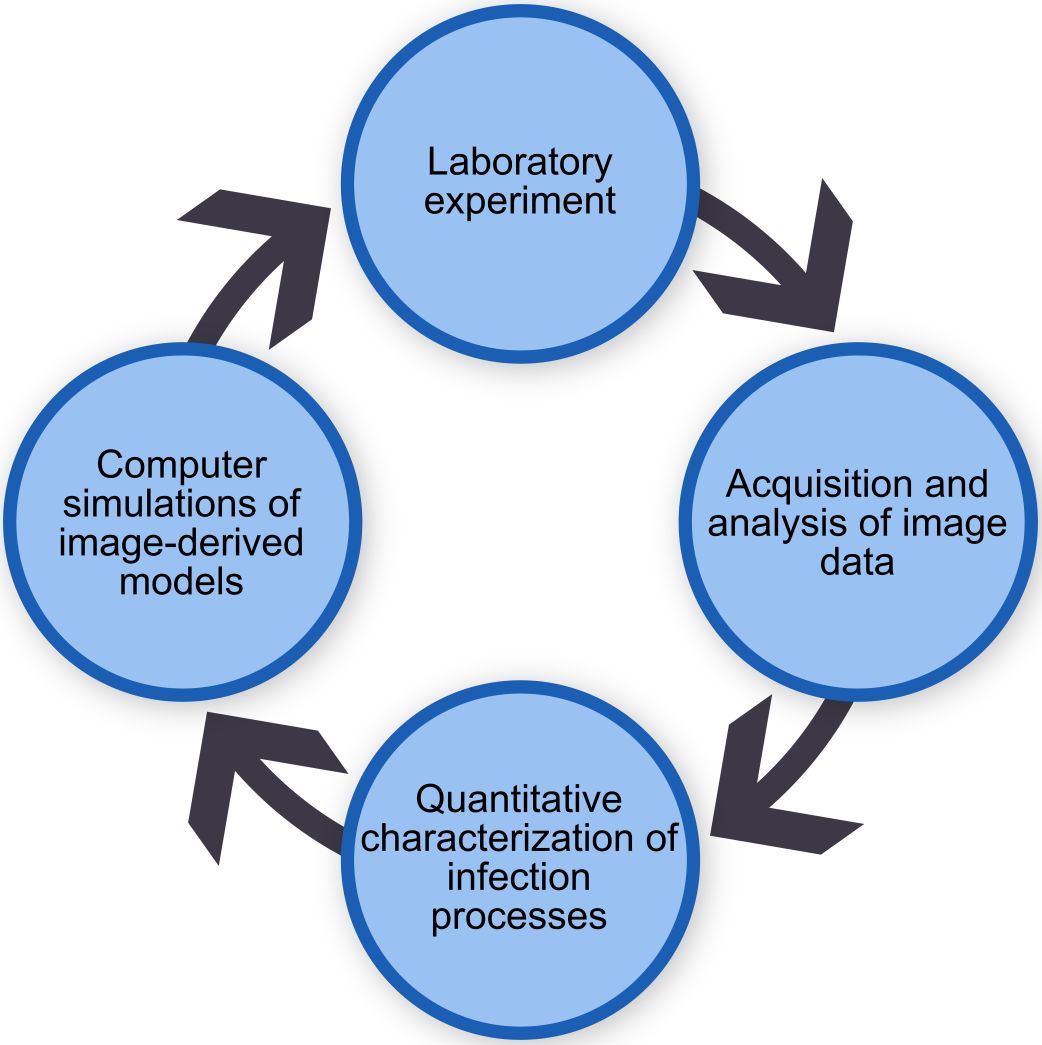


Figure 1.1: Image-based systems biology approach. A laboratory experiment is set up with the purpose of capturing characteristic interactions via images. These are analyzed with computer programs to obtain quantitative information about the visualized processes. A detailed insight is given by the implementation of computer models that are built on the knowledge gained from the images. *In silico* experiments based on the models lead to the generation of new hypotheses that are evaluated by the implementation of new laboratory experiments. Adapted from [93, Figure 1].

1.1 Host-pathogen interactions

To understand the mechanisms of host-pathogen interactions, these terms first need to be defined. The simplest to explain is "host", which refers to the organism that interacts with microorganisms [26, 25]. The latter are not necessarily pathogens and can alternatively form commensal or even symbiotic relationships with the host. An organism is only referred to as pathogen, if it causes significant damage to the host with the result of its inability to maintain homeostasis [26]. In the context of immune reactions this means that the immune system is perturbed to a level that generates a response [30, Chapter 1, Page 9]. Damage is characterized by interruptions in the structure or function of affected areas and can happen on multiple scales: a small-scale harm on a single-cell level, medium-scale damage affecting a region of tissue, and the large-scale loss of organ function. For example, cells can be damaged by necrosis, apoptosis, loss of signal transporter functions, or malignant transformations. Tissues can be affected by tumors, fibrosis, or inflammation, while whole organs can lose their function if transport ducts are blocked, or due to the loss of functional cells, e.g., glomeruli in a kidney [73].

The two major mechanisms behind pathogen-induced damage are toxins and harm that is introduced by the immune system. Various microorganisms produce toxic agents responsible for the clinical symptoms [30, Chapter 21, Page 334]. Examples for toxin-producing microorganisms are *Corynebacterium diphtheriae*, *Vibrio cholerae*, *Clostridium tetani*, or Shiga-toxin producing *Escherichia coli* [73]. All toxins have in common that even a minuscule amount is sufficient to induce the detrimental effect to the host. The mechanisms behind the toxicity vary greatly and include the induction of muscle spasms by acting as exciting agents, change of water uptake in intestinal cells, or specific damage to immune cells. For example, *Bacillus anthracis* toxin damages the expression of anti-apoptotic genes in certain immune cells, thus causing their death.

The immune system itself can be the source of the damage by responding inappropriately, i.e., the immune system is too inactive or responds too aggressively. This is the case for sepsis [11], where a local infection caused by a microorganism escalates into a systemic inflammation of the whole organism due to regulatory issues in the immune system. As inflammation interferes with the normal function, e.g., by causing hypoxia, sepsis can lead to a failure of multiple organs.

The interaction between a host and a microorganism [25] starts with their contact (see Figure 1.2, blue box), followed by an infection, which is characterized by the multiplication of the microorganism inside the host. Depending on the state of the host and the properties of the microorganism, the infection can evolve into one of three conditions characterized by their damage to the host: if the host is not damaged over time, the microorganism is a commensal; a colonization is characterized by a linear damage over time; finally, if the host is damaged exponentially over time, the interaction is referred to as "disease" that can kill the host if eradication is unsuccessful. The complexity of host-pathogen interactions is highlighted by their dynamic nature, meaning that the interactions can transition based on the state of the host or microorganism. For example, it is possible for a commensalist to transform into a pathogen, as showcased by the fungus *Candida albicans* which

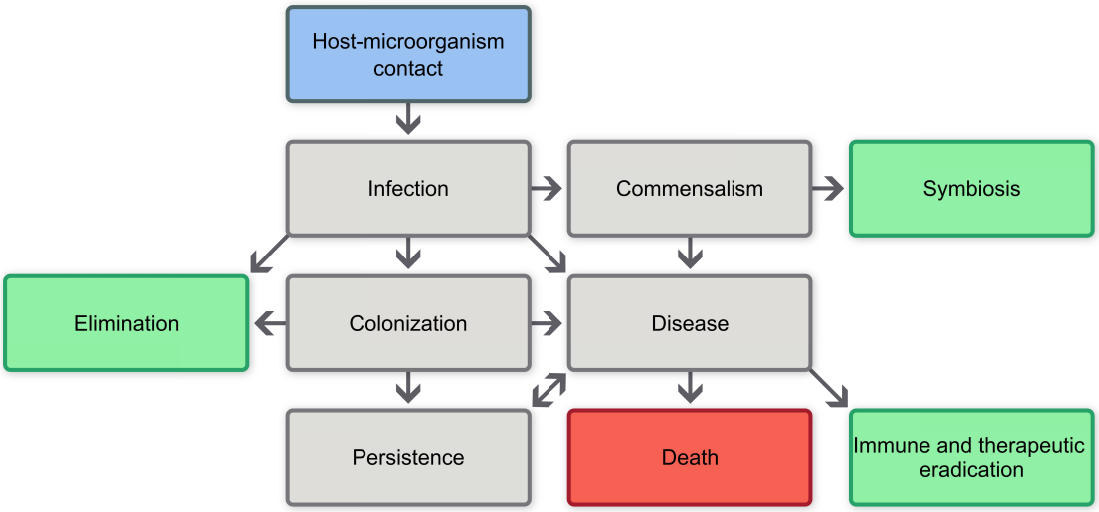


Figure 1.2: Interactions between hosts and microorganisms. The starting point is the contact between host and microorganism (blue box), leading to an infection. Via transitioning (arrows) between different condition types, the outcome can be either positive for the host (green boxes), or end in its death (red box). Host-pathogen interactions are dynamic and can shift between multiple intermediary states (gray boxes). Adapted from [26, Figure 1].

is known to transition from a commensal into a pathogen if the immune system is compromised [62]. Another possibility is that a commensalism is converted into a mutualism where both the host and pathogen greatly benefit from the interaction. Colonies can develop into diseases, or transition to a chronic infection where the damage over time is only applied on a constant level. Eradicating the microorganism from the host is possible during the infection, colonization, and disease phases and involves physical barriers, elimination via the existing flora of the host, externally applied treatments, or the immune system.

1.1.1 Endocytosis and phagocytosis

A defense against pathogens is provided by the host's immune system. In its simplest form, it is based around the ingestion of pathogens by specialized cells. In the human immune system, this includes macrophages, polymorphonuclear leukocytes, and dendritic cells [30, Chapter 2, Page 12-14]. Ingesting molecules or macro-particles into a cell is called "endocytosis" [30, Chapter 2, Page 19] that is further differentiated into "pinocytosis" and "receptor-mediated endocytosis". The former targets very small particles or liquids unspecifically [30, Page 370], while the latter is characterized by the ingestion of specific particles.

Pinocytosis is an unspecific ingestion that forms an invagination at the plasma membrane with the purpose of encompassing particles or molecules in a close vicinity to the membrane. This is followed by the closing of the membrane, leading to the formation of a vesicle, i.e., a compartment within the cell. These are also produced by the specific mode of ingestion with the difference that the invagination

only occurs if particle of interest binds to an array of receptor molecules that are present on the cell membrane. After being incorporated into the cell, vesicles are merged with an existing acidic compartment called "endosome". The environment created by this incorporation is necessary for the function of nucleases, lipases, and proteases obtained from the absorption of a lysosome. The result is the degradation of the ingested particles into molecules useful for cells, e.g., nucleotides, sugars, and peptides.

The ingestion of invading particles is called "phagocytosis" and targets, for example, bacteria or viruses [30, Page 19, Page 58]. Similar to a receptor-mediated endocytosis, the phagocytic cell can recognize molecules specific to the pathogen. It is also possible that the particle is marked with opsonins that can consist of antibodies and other pathogen-recognizing immune system components. The particle is engulfed into a compartment termed "phagosome" that is merged with lysosomes to introduce ingesting enzymes. As these usually cannot penetrate the physical barriers around the particle, the newly formed phagolysosome produces cytotoxic agents that include nitric oxide, hypochlorous acid, or reactive oxygen species (ROS).

Many pathogens adapted to such immune system responses. For example, spores of the fungus *Aspergillus fumigatus* produce types of melanin that protects the cell against ROS [57]. Another example of an unsuccessful elimination of a pathogen is the non-lytic expulsion of *Candida albicans* spores [8] that involves the raise in pH levels within the phagolysosome. Similar strategies are applied by the bacterium *Mycobacterium tuberculosis* that can survive phagocytosis events and keeps surviving inside the host cell, thus avoid elimination by the immune system and drug treatments [112]. These examples highlight the need for understanding the full set of interactions between hosts and pathogens, as these can lead to the development of more effective treatments for live-threatening conditions. With the continuing development of imaging technologies, it is possible to visualize the involved cells and derive the mechanisms of host-pathogen interactions via quantitative measurements.

1.2 Imaging techniques

In the following section, we will briefly describe commonly utilized methods for imaging host-pathogen interactions. The first technique that will be presented is transmitted light (TL) microscopy which is based around focusing light through the specimen to capture its shadow (see section 1.2.1). To generate images with higher contrast, methods based on fluorescence were developed. One approach is confocal microscopy that is capable of visualizing specimens in three dimensions via optical sectioning (see section 1.2.2), although with limitations on the depth especially in highly scattering tissues. We thus also introduce light sheet fluorescence microscopy (LSFM) which illuminates the specimen via a sheet of laser light from a direction orthogonal to the optical axis with the purpose of avoiding light scattering (see section 1.2.3). Finally, we will present multispectral optoacoustic tomography (MSOT) that exploits the optoacoustic effect to capture pharmacoki-

netics within whole organs *in vivo* via laser-induced ultrasonic sound waves (see section 1.2.4).

1.2.1 Transmitted light microscopy

The most straightforward method to image the interactions of cells is a transmitted light (TL) microscope, which focuses light through the object of interest and a set of magnifying lenses [118, Chapter 4.1, Page 75-78]. The basic components of such a microscope are a light source, collector and condenser lenses, the specimen to be imaged, and the objective and eyepiece lenses (see Figure 1.3). Light is generated by an incandescent or halogen-based lamp, or a light-emitting diode (LED). Due to the unevenness of this light, a set of lenses and diaphragms are used to collect and condense the light into a parallel or direct beam. The light is then transmitted trough the specimen to be magnified by the objective lens, which comprises a set of four to six elements. Finally, the light is focused through the eyepiece and reaches a camera or the human eye.

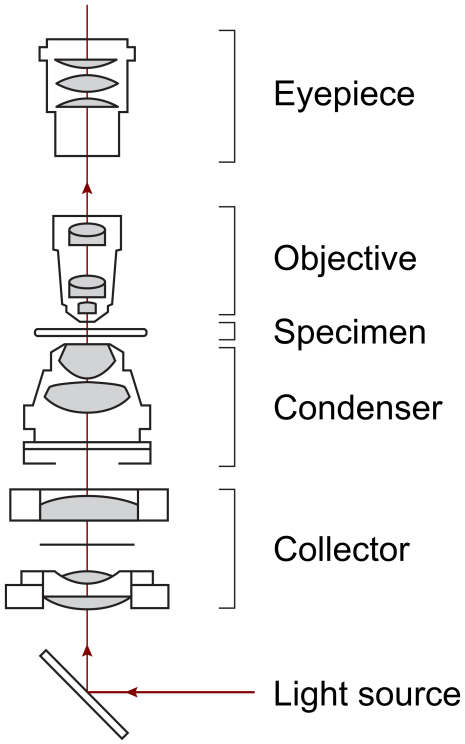


Figure 1.3: Transmitted light microscope. Light (red line) generated by a light source passes through the collector and condenser to achieve even illumination. After reaching the specimen, magnification is applied in the objective. Finally, the light reaches the eyepiece. Adapted from [118, Figure 4.1].

Given this simple and easily extendable structure, a multitude of variants have been developed that that use different techniques for distinguishing the object of interest from the background. Examples include dark-field microscopy [118, Chapter 11.4.2, Page 252-255], phase contrast microscopy [118, Chapter 11.12, Page

264-270], and differential interference contrast microscopy [118, Chapter 11.14, Page 273-278].

1.2.2 Confocal microscopy

TL microscopy and its variants are limited in their ability to generate contrast due to their reliance on external illumination. For example, bright field microscopy only allows contrast improvements by up to 40 % [118, Chapter 15.4, Page 388-389]. Another disadvantage is the difficulty of differentiating structures with similar optical properties. A solution is to make structures of interest emit light by themselves. This can be achieved by labeling these with fluorophores [118, Chapter 16.1, Page 405-409], which are molecules that generate light of a specific wavelength shortly ($10^{-9}s$ to $10^{-12}s$) after being excited with light of another specific wavelength. This process is referred to as "fluorescence" and allows precise targeting of structures and even molecules by controlling the excitation wavelengths [118, Chapter 15.3, Page 389]. As the excitation and emission wavelengths are highly specific, multiple structures can be stained and recorded, making it possible to distinguish multiple characteristics of an interaction within the same experiment.

These achievements rely on the presence of the appropriate dye. It is possible to introduce these by providing pre-made fluorophores. Alternatively, the organism can be genetically modified to synthesize fluorescent proteins, for example, green fluorescent protein (GFP) [118, Chapter 16.3, Page 409-411]. Depending on the targeted structure, it is even possible to avoid the artificial introduction of dyes or genes and target a variety of natural fluorophores present in many cells [118, Chapter 16.7, Page 417].

Challenges of fluorescence microscopy include the distinction of overlapping emission frequencies of certain fluorophores, photobleaching that leads to gradual loss of their emission functionality [118, Chapter 15.3, Page 389], and the capture of all emissions regardless of their depth. This last property leads to the blurring of signal outside the focal plane of origin [118, Chapter 17.1, Page 426], resulting in a limitation on the achievable contrast. As this effect is related to depth, specimens of a certain thickness cannot be sufficiently visualized.

A solution to blurring is to exclude light emitted outside the current focal plane via optical sectioning. An implementation can be found in confocal microscopes, where pinholes restrict the illumination to a specific location [118, Chapter 17.3, Page 427-430]. The components of a confocal microscope (see Figure 1.4) include one or multiple gas or LED laser light sources that emit the excitation frequencies, one or multiple pinholes for focusing the excitation and emission light, a set of mirrors and lenses to control the location of the illumination, and photomultiplier tube for capturing the emitted light. The targeted excitation is achieved by a set of movable mirrors that scan the specimen in a raster pattern, controlling the point where the fluorophores should be excited. To suppress the blurring in the depth axis, the laser light is focused to ensure that the maximum intensity is at the specified depth.

The disadvantages of such a raster-scanning microscope include the increased pho-

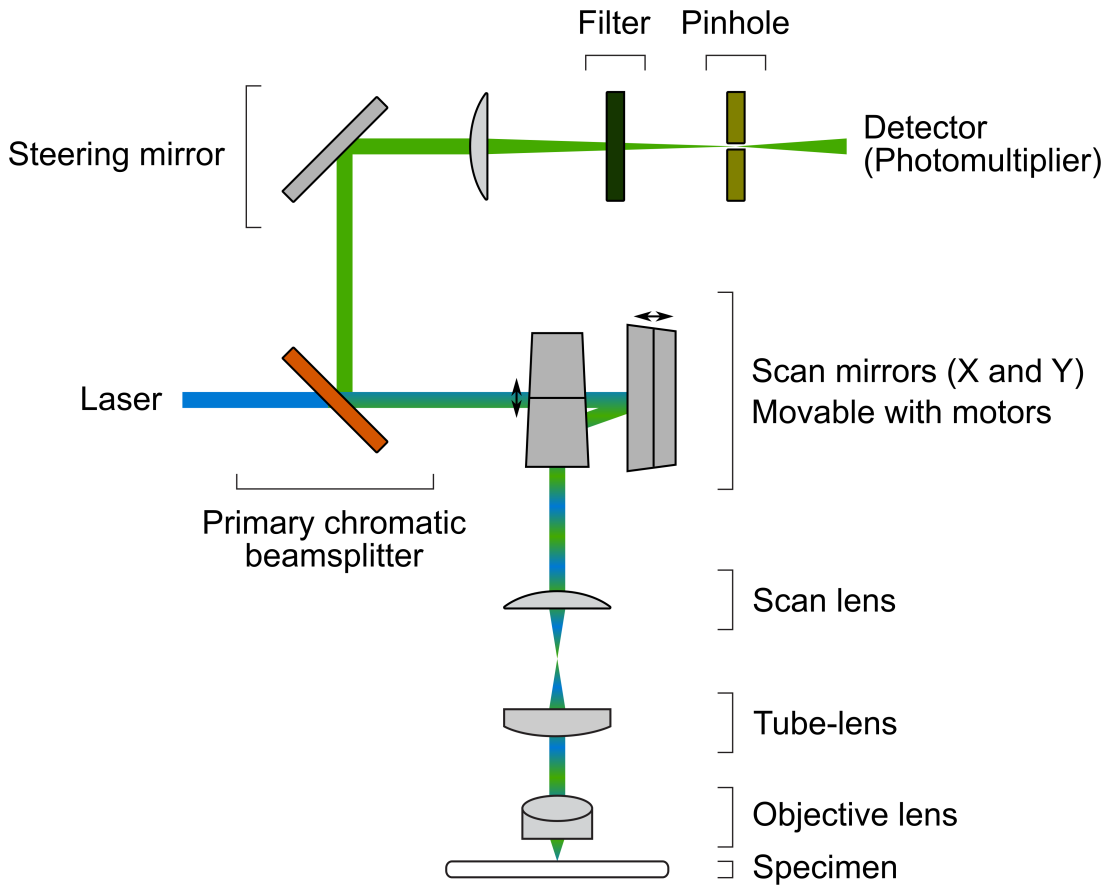


Figure 1.4: Components of a confocal microscope. Excitation laser light (blue) passes through the primary chromatic beamsplitter and a set of movable mirrors that allow to target a specific location in the X and Y axis. Afterwards, it continues through a scan lens, tube-lens, and objective to excite fluorophores in the specimen. The resulting emission light (green) passes through the objective, tube-lens, scan-lens, and mirrors, and is reflected by a beamsplitter. Here, it is reflected by a steering mirror and is focused through a lens and the pinhole to be captured by a photomultiplier tube. Adapted from [118, Figure 17.2b].

tobleaching due to the use of lasers, and the slow image generation caused by the point-by-point buildup of the image, thus severely limiting the achievable time resolution. These issues are partially resolved in the "spinning disk confocal microscope" that is based around the principle of illuminating multiple points in parallel. This is achieved by replacing several components: the singular illumination lens is replaced with a disk that contains an array of micro lenses, thus generating multipoint illumination (see Figure 1.5). The same principle is applied to the illumination pinhole by replacing it with another disk containing holes that are corresponding to the micro lenses [118, Chapter 17.6, Page 437-439].

These improvements, however, do not address another disadvantage of confocal microscopy: both the excitation and emission light travel through the same optical axis [118, Chapter 19.1, Page 485]. This greatly reduces the brightness of the resulting excitation light if the specimen is of sufficient thickness or consists of

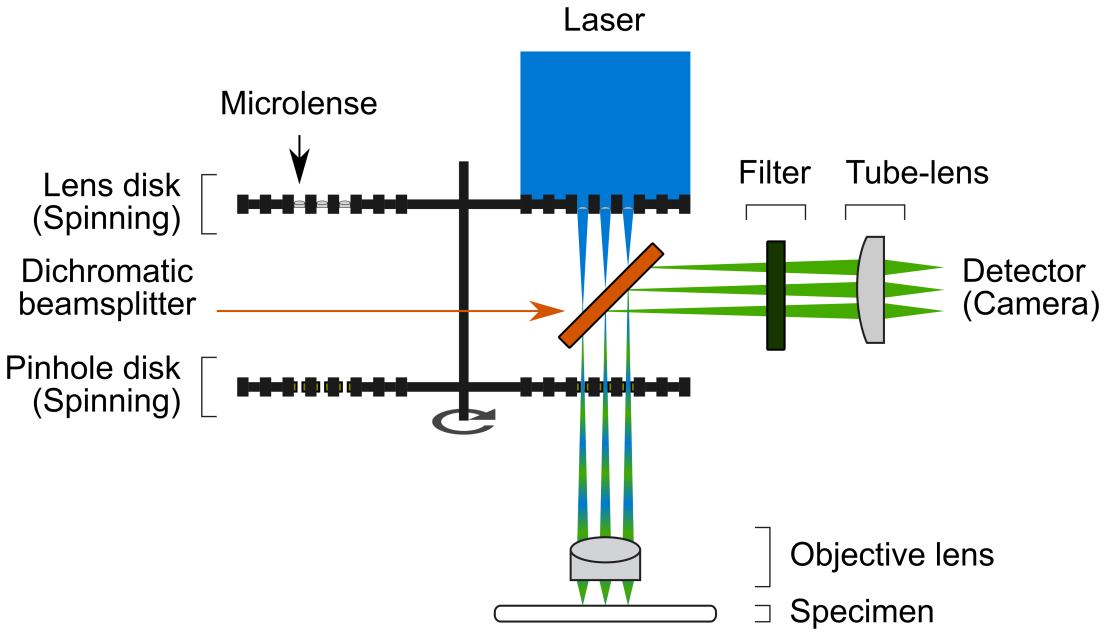


Figure 1.5: Components of a spinning disk confocal microscope. Excitation laser light (blue) passes through micro lenses within a spinning disk, a dichromatic beamsplitter, the second spinning disk providing the appropriate pinhole. The light continues through the objective lens to the specimen, where it excites the fluorophores. The resulting emissions (green) pass through the objective and pinhole, but are reflected by the mirror, where it is filtered and captured by a camera after being focused via a tube-lens. Adapted from [118, Figure 17.9].

highly scattering tissue. While a compensation by an increase of the laser light intensity is possible, this strategy must be weighed against the effects of increased photobleaching.

1.2.3 Light sheet microscopy

Light sheet fluorescence microscopy (LSFM) addresses the disadvantages of confocal microscopy (see section 1.2.2) by illuminating the sample from the side [118, Chapter 19.1, Page 484-486]. This light is usually an approximately 1 μm to 10 μm thin sheet of laser light that illuminates a whole plane at once. To capture the entire three-dimensional (3D) object, the sheet is moved across the whole depth of the specimen while capturing the emitted light with a camera. Later, computer software is utilized to combine these individual planes into a 3D image (see Figure 1.7).

A simple light sheet fluorescence microscopy setup consists of two axes that are responsible for illumination and detection (see Figure 1.6), respectively. To illuminate the specimen, laser light passes through a set of lenses that are responsible for creating the light sheet. These include a beam expander lens, a cylindrical lens, and a tube-lens. The detection axis is perpendicular to the illumination axis. Here, the emitted light is focused through an objective lens to be captured by a

camera. As a whole image plane is captured in one step, a high time resolution can be achieved without running into issues with photobleaching. For example, it is possible to capture a time series of whole organisms in 3D [135] or track embryonic cells during their development [134]. The basic setup presented here can be extended by, for example, adding components for rotating the sample or light source with the purpose of improving the illumination, and inclusion of multiple light sources and cameras to capture multiple fluorophores.

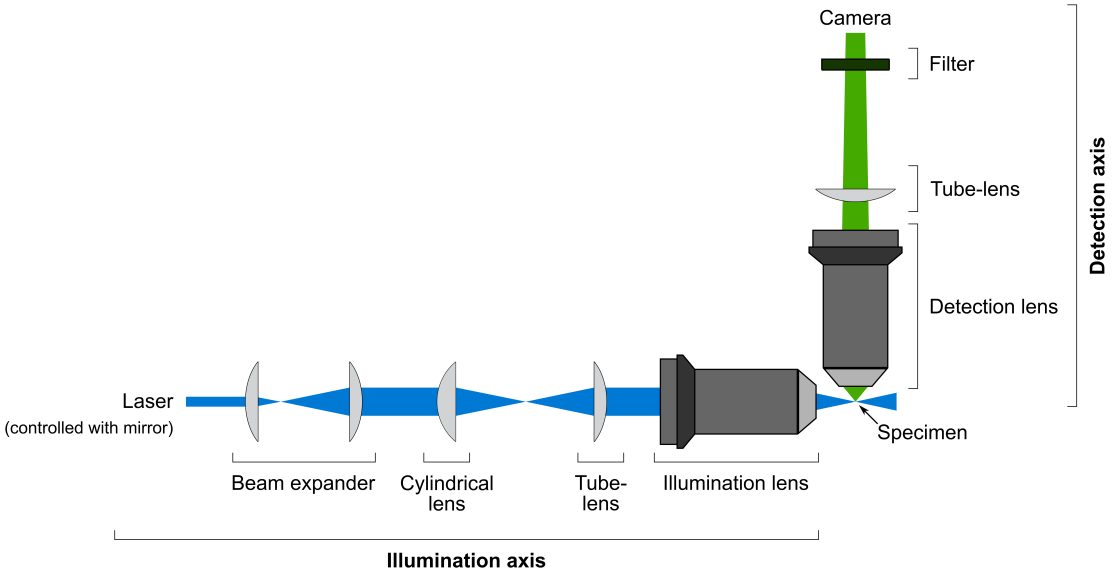
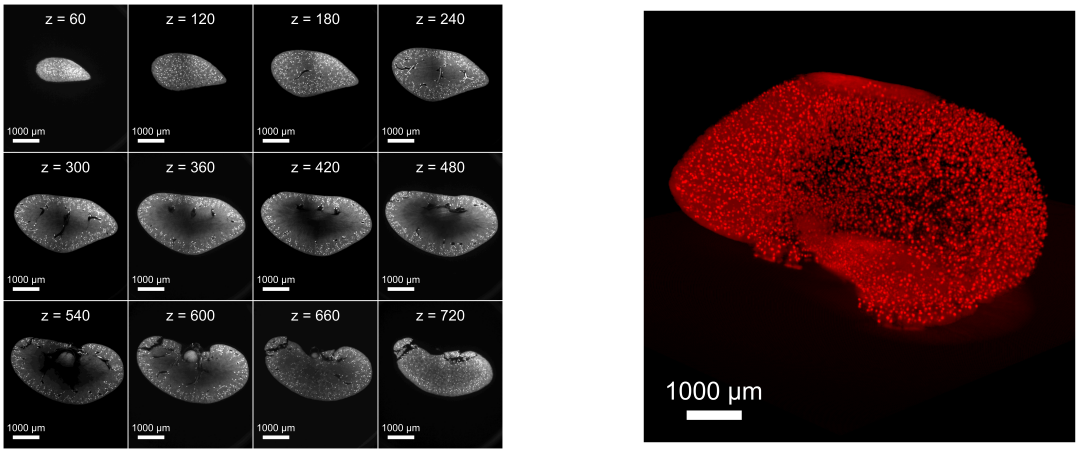


Figure 1.6: Components of a light sheet microscope. The microscope can be roughly divided into an axis for illuminating the fluorophores and one for detecting the emitted light. Excitation laser light (blue) is controlled with a mirror and passes through a beam expander, cylindrical lens, tube-lens, and illumination objective lens to form a sheet of light that excites one plane of the specimen. The resulting emission light (green) passes through the detection objective lens, a tube-lens, and filter for the excitation wavelength to be captured by a camera. Adapted from [118, Figure 19.2a].

A drawback of light sheet fluorescence microscopy is that specimens are required to be translucent, which makes the application of clearing protocols necessary [118, Chapter 19.7, Page 497-498]. As the function of organ tissue is destroyed by these, light sheet fluorescence microscopy cannot be utilized to analyze organs *in vivo*, unless they are translucent to begin with. If this is not the case, alternative imaging methods can be applied instead.

1.2.4 Multispectral optoacoustic tomography

An alternative to microscopy that is capable of visualizing the function of whole organs *in vivo* is multispectral optoacoustic tomography (MSOT) [114]. The basic principle behind this technique is the optoacoustic effect: if tissue absorbs light, sound waves are generated due to thermoelastic expansion. Consequently, nanosecond pulses of nano-joule laser light lead to the creation of ultrasound waves



(a) **(b)**

Figure 1.7: light sheet fluorescence microscopy image of a whole murine kidney. The data was obtained from an existing study [73] and shows a whole murine kidney with glomeruli visible inside (bright dots). The image stack consists of 784 slices. **(a)** Selection of two-dimensional (2D) image slices from different depths. **(b)** Render of the whole 3D image. A black-to-red look-up-table was applied.

that can be captured by multiple detectors. With MSOT, the sample is illuminated sequentially with multiple wavelengths [128] that can be distinguished by a computer software via spectral unmixing. Due to the fast recording time, MSOT records the presence of specific fluorophores as 3D time series, thus allowing to infer the pharmacokinetics from the resulting four-dimensional (4D) image.

The setup of an MSOT experiment involves a laser light source, and ultrasonic detectors based on interferometry or the piezoelectric effect (see Figure 1.8). To allow generated ultrasound waves to reach these detectors, the specimen is put into gel or water. The captured signals are multiple one-dimensional (1D) sound recordings. To obtain an image, a computer program must be used [114] to solve two problems: the first is the "acoustic inverse problem", where it must be estimated how much energy was put into the specimen at the specified point. Afterwards, the computer must find a transformation to convert multiple sound signals into an image. This is also referred to as "optic inverse problem". Advanced algorithms were developed to solve these problems by implementing various strategies, for example, by the conversion of the signal into the frequency domain, application of time-reversal, or the utilization of computer models [114].

A benefit of MSOT is the ability to illuminate a specimen with multiple wavelengths. As the method is not able to resolve the molecular level, each voxel contains signals from multiple molecules [133]. These need to be unmixed to obtain intensities for each targeted molecule. This is a challenging task due to spectral features changing in deeper tissue regions, and the presence of intrinsic chromophores that include blood, lipids, or melanin. There are multiple algorithms available that, for example, attempt a spectral unmixing by employing linear or non-linear mixture models to find all involved components [133]. Independent of

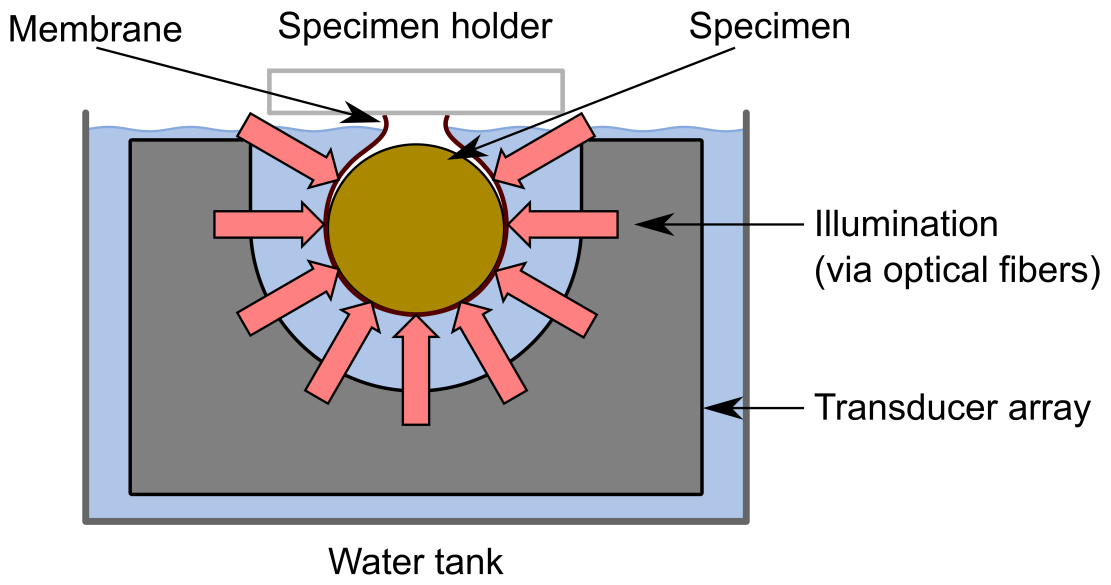


Figure 1.8: Components of a multispectral optoacoustic tomography setup. The specimen is held within a water tank via a holding mechanism and a plastic membrane. It is evenly illuminated via optical fibers with short pulses of laser light. The resulting ultrasonic waves are detected in an array of acoustic transducers. This setup was designed to image mice. Adapted from [22, Figure 1a].

the chosen algorithms, computer tools then produce images with three spatial and one time dimension with multiple channels.

1.3 Automated image analysis

In the previous sections, we introduced various methods to capture the interactions present in a wet lab experiment as images. According to the principles of image-based systems biology (IbSB), the next step is to quantify the data. A straightforward approach is to analyze the images manually, which is time consuming and error-prone due to researcher bias. An alternative is to create a computer program that applies all necessary steps automatically, which makes it possible to reproducibly process thousands of images in a short time. In section 1.3.1, we will first give a brief explanation on how images are represented inside computers, followed by establishing the foundations of many image processing operations in form of spatial filters (see section 1.3.2). Afterwards, we introduce various classes of image operators that are commonly applied in automated image analysis: as the captures produced by microscopes are affected by various scattering and noise processes, we establish the image degradation model and how it is utilized to revert the aforementioned processes (see section 1.3.3). The restored image is the subject of section 1.3.4 that introduces segmentation operators based on intensity thresholding for the extraction of regions of interest (ROIs). These are further processed by morphological operations with the purpose of applying corrections

(see section 1.3.5). Finally, we will briefly introduce classification as generalization of segmentation (see section 1.3.6), as well as deep learning as a method that implements supervised machine learning (see section 1.3.7).

1.3.1 Image representation in computers

Computers represent images as arrays of addressable square units called "pixels" [118, Chapter 30.3, Page 710-711]. Each pixel is assigned to a fixed space coordinate and given an intensity value. As computers have finite memory, there is a limit on the number of assignable intensity levels L, which is given by $L = 2^k$, where $k > 0$ denotes the number of bits, i.e., the smallest unit that can be stored in a computer [51, Section 2.4.2, Page 55-57]. Each bit stores a binary digit, meaning that it can only represent a zero or one. As with the decimal system, multiple digits can be combined into a number. To calculate the decimal value $(x)_{10}$ of a binary number $(x)_2$ with k positions, the value $(x)_2\{i\}$ of each position $0 > i \geq k$ is multiplied with 2^{i-1}. The resulting values are summarized:

$$(x)_{10} = \sum_{i=1}^{k} 2^{i-1} \cdot (x)_2\{i\}. \tag{1.3.1.1}$$

For example, the binary number $(10111)_2$ is evaluated to the decimal number $(23)_{10}$. We note that the position i here is evaluated from right to left, i.e., the right-most digit is at the first position.

As only a limited number of bits is available, light levels generated by sensors must be quantized into appropriate discrete approximations via an analog-digital converter. Choosing an appropriate bit depth is thus essential to ensure a correct representation of the information gained by an imaging method [118, Chapter 30.6, Page 715].

Given all pixels were assigned a location and value, an image is formed, which from a mathematical standpoint is a function $f(x, y)$, where x and y are the spatial coordinates [51, Section 1.1, Page 1]. In practice, it is represented as 2D matrix with M rows and N columns, meaning that x and y are integral numbers. As the size is fixed, the memory usage of an image can be calculated as $M \cdot N \cdot k$ bits. Given this representation of an image, computer programs can be developed that apply various operations on regions of pixels.

1.3.2 Image transformations and spatial filtering

Automated image analysis is based on transformation of pixel values to remove noise, enhance features, and to extract and quantify structures of interest. Here we again will focus only on grayscale images.

The simplest operation is applied to each individual pixel is a transformation $s = T(z)$ that yields a new pixel value s for an input pixel value z [51, Section 2.6.5, Page 85]. Applying such an operation to each pixel results in a new image

$g(x, y) = T(f(x, y))$. An example for a single pixel transformation is the addition of a constant value c to each pixel:

$$T_{\text{add constant}}(z) = z + c. \tag{1.3.2.1}$$

Linear spatial filtering A more advanced operation involves neighboring points around each pixel, and is also referred to as "spatial filtering" [51, Section 2.6.5, Page 85]. This neighborhood is rectangular with a size $m \cdot n$ and centered around the currently processed pixel location (x, y). The neighborhood can also be characterized by two parameters $a > 0$ and $b > 0$ that indicate the extent of the neighborhood around the center point. An example for a spatial filter is the local average

$$g_{\text{local average}}(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} \frac{1}{m \cdot n} \cdot f(x + s, y + b) \tag{1.3.2.2}$$

that summarizes all values in the neighborhood and weights the result by dividing it by the number of neighborhood pixels.

The local average applies a fixed weight $\frac{1}{m \cdot n}$ to each neighborhood value, which opens the possibility for a generalization: instead of a predefined value, the weights are sourced from a function $w(s, t)$, where s and t are relative to the current center point (x, y):

$$w(x, y) \circ f(x, y) = g(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) \cdot f(x + s, y + t). \tag{1.3.2.3}$$

The resulting function is termed "linear spatial filter", due to the linearity of its underlying mathematical operation [51, Section 3.4.1, Page 145].

An interesting observation can be made regarding the weight function: both of its parameters are effectively discrete and limited to the extents of the neighborhood, i.e., $-a \leq s \leq a$ and $-b \leq t \leq b$. Thus, $w(s, t)$ can be represented as matrix with $2 \cdot a + 1$ columns and $2 \cdot b + 1$ rows where elements are accessed relative to its center. Such a matrix is also called a "filter matrix". An example of a local average filter with extents $a = 1$ and $b = 1$ is

$$w_{\text{local average}}(s, t) = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}. \tag{1.3.2.4}$$

Convolution The type of linear spatial filter defined in eq. (1.3.2.3) is also referred to as "correlation" [51, Section 3.4.2, Page 146-150]. A closely related group of linear spatial filters is based on "convolution" and differs by rotating the filter matrix by 180 °:

$$w(x, y) \star f(x, y) = \sum_{s=-a}^{a} \sum_{-b}^{b} w(s, t) \cdot f(x - s, y - t). \tag{1.3.2.5}$$

Symmetric filter kernels consequently fulfill both the properties of being a correlation and convolution. If a non-symmetric filter is given, it is generally preferred to apply a convolution. The reasoning behind this rule will be briefly explained in the following part.

Let there be an image $f(x, y)$ that has a value of one at its center and zero everywhere else, also referred to as a "discrete unit impulse"

$$f(x,y) = \begin{cases} 1 & \text{if } x = \frac{M}{2} \wedge y = \frac{N}{2} \\ 0 & \text{otherwise} \end{cases}. \tag{1.3.2.6}$$

Let there also be a non-symmetric filter kernel $w(s, t)$, for example,

$$w(s,t) = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}. \tag{1.3.2.7}$$

The correlation $w(x, y) \circ f(x, y)$ results in a copy of the filter matrix, but rotated by 180 °:

$$w(x,y) \circ f(x,y) = \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}. \tag{1.3.2.8}$$

A convolution $w(x, y) \star f(x, y)$ on the other hand yields an unchanged copy of the kernel:

$$w(x,y) \star f(x,y) = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}. \tag{1.3.2.9}$$

Thus, by utilizing a convolution, a copy of a function at the location of the impulse is returned, which is an essential requirement for allowing filtering in Fourier space [51, Chapter 4.2.3, Page 204].

1.3.3 Image enhancement

Two common problems that are faced during the analysis of images are noise and degradation. These processes are caused by the generation of the images and for example, include the capture of photons outside the selected focal plane in fluorescence microscopes (see section 1.2.2). The consequence is a great influence on various image processing steps, including edge detection [51, Section 10.2.4, Page 704], and thresholding [51, Section 10.3.4, Page 704].

A common model for the relationship of degradation and noise assumes that the true image $f(x, y)$ is convolved with a degradation function $h(x, y)$, while noise $\eta(x, y)$ is added to the convolved image:

$$g(x,y) = h(x,y) \star f(x,y) + \eta(x,y). \tag{1.3.3.1}$$

The cause and properties of $h(x, y)$ greatly depend on the image source and capturing methods. For example, atmospheric turbulence in landscape images [51, Section 5.6, Figure 5.25, Page 348], blurring in fluorescence microscopy (see section 1.2.2), or tissue scattering in confocal microscopy and light sheet fluorescence microscopy (see section 1.2.3).

Given the degradation model in eq. (1.3.3.1) and the properties of convolutions in Fourier space, it is possible to derive methods to revert the degradation.

Operations that remove the convolutional image degradation processes are referred to as "deconvolution". A naive deconvolution algorithm is "inverse filtering" [51, Section 5.7, Page 351], which is based on the fact that convolution in the spatial domain is equivalent to a multiplication in Fourier space [51, Section 4.2.5, Page 210]. As the inverse of a multiplication is a division, the original Fourier space representation of the true image $\hat{F}(u, v)$ in Fourier space can be reconstructed by dividing the degraded image $G(u, v)$ by the degradation function $H(u, v)$:

$$\hat{F}(u, v) = \frac{G(u, v)}{H(u, v)}. \tag{1.3.3.2}$$

The spatial domain then can be obtained by applying an inverse Fourier transform [51, Section 4.4.1 and 4.5.5, Pages 221-222 and 235-236].

A disadvantage of the naive method is that it assumes that the exact parameters of the degradation are known [51, Section 5.6, Page 346] and no noise is present, which is not the case in microscopy images. Advanced methods thus attempt to correct for both noise and degradation. An example is the Wiener filter [51, Section 5.8, Page 352-355] that estimates the true image by incorporating both the properties of the degradation function and the statistical nature of noise: the method finds a filter that minimizes the mean square error between the estimation of the true signal and the degraded image.

An alternative approach for deconvolution makes use of the point spread function (PSF) that describes how a single point of light is scattered within the specimen. The PSF can be utilized to re-assign photons back to their original location [118, Chapter 21.9, Page 537-538]. These methods repeat two operations until the solution converges: first, the object locations are estimated based on the properties of the PSF; second, algorithms for constrained noise removal are applied. Consequently, the approach is referred to as "constrained iterative deconvolution".

The PSF can be roughly estimated from the microscope parameters, or measured by imaging a bead smaller than the resolution limit [118, Chapter 21.6, Page 530]. Here we note that it is also beneficial to measure PSF within multiple depths to correct for different scattering behavior in deeper tissues.

While advanced deconvolution methods attempt to remove noise, the success can greatly vary based on the algorithm, its parameters, and the properties of the data [92]. Thus, it is common that additional methods for the removal of the remaining noise, and enhancement of contrast are applied. For example, an image can be processed with a local average filter [51, Section 5.3.1, Page 322] that estimates the true pixel value based on the surrounding values, while low-contrast structures are enhanced with contrast limited adaptive histogram equalization

(CLAHE) [108]. The resulting image then can be further processed to extract and quantify the visible structures.

1.3.4 Segmentation

The techniques introduced in section 1.3.3 improve images by suppressing degradation and noise, and enhancing the contrast. At this state, the computer program is not aware of which pixels are assigned to the structures of interest or to the background. The process that generates this differentiation is called "segmentation" and divides the image into one or multiple ROIs [51, Chapter 10, Page 689].

An intuitive segmentation method is based around an intensity threshold that changes the image intensities to predefined values that denote a classification into foreground or background [51, Section 10.3.1, Page 738-739]. The decision depends on whether the intensity of the pixel is larger than a threshold value T and results in a binary image

$$g(x,y) = \begin{cases} 1 & \text{if } f(x,y) > T \\ 0 & \text{if } f(x,y) \leq T \end{cases} \tag{1.3.4.1}$$

where a non-zero value indicates that the pixel is part of a structure.

The challenge of intensity thresholding is that a suitable T must be determined. One option is to choose the threshold manually, which can be slow and difficult to reproduce due to the manual assessment by an expert. An alternative is to automatically calculate a threshold based on statistical properties of the image. There exist multiple methods that achieve this; for example, there are operations based on entropy [61, 69, 143], preservation of image moments [132], or functions that consider geometric properties of the histogram [144]. A popular thresholding operation is Otsu's method [51, Section 10.3.3, Page 742-747] that is based on the principle that the difference between foreground and background pixels should be maximized. We will briefly explain Otsu's method in the following paragraphs.

This inter-class variance σ_B^2 is measured for each possible threshold k, given the mean intensity of pixels assigned to the background $m_1(k)$ and foreground $m_2(k)$, the mean pixel value m_G, the probability $P_1(k)$ of choosing the background pixel randomly, and its inverse $P_2(k) = 1 - P_1(k)$:

$$\sigma_B^2 = P_1(m_1(k) - m_G)^2 + P_2(m_2(k) - m_G)^2. \tag{1.3.4.2}$$

Based on the histogram of the pixel values $h(r_k)$, the probability $P_1(k)$ can be obtained by calculating the normalized cumulative histogram of the pixel intensities:

$$P_1(k) = \sum_{i=0}^{k} \left[\frac{1}{M \cdot N} \cdot h(r_i) \right]. \tag{1.3.4.3}$$

Otsu's method then calculates the optimal threshold T_{Otsu} by finding a k that maximizes σ_B^2:

$$T_{Otsu} = \underset{k \in \{0,1,\ldots,L-1\}}{\arg\max} (\sigma_B^2). \tag{1.3.4.4}$$

The resulting value is then utilized in eq. (1.3.4.1) to binarize the image.

1.3.5 Morphological processing

The result of various segmentation methods is an image that contains ROIs, which are usually represented by predefined pixel values (see section 1.3.4). In the case that the results are not sufficient, operations for processing the shapes of objects can be utilized to correct segmentation mistakes [51, Chapter 9, Page 627]. Examples for such morphological operations are the filling of holes [51, Section 9.5.2, Page 643], extraction of connected components from a pixel image [51, Section 9.5.3, Page 645], and skeletonization of segmented areas into lines [51, Section 9.5.7, Page 651-653].

A family of morphological operations is based on two fundamental functions termed "erosion" and "dilation" [51, Section 9.2, Page 630]. We note that while these can be described with set theory [51, Section 9.1, Page 628-629], we will focus on an alternative representation based on local rank operations.

Morphological operations are spatial filters (see section 1.3.2) where the neighborhood is defined by a "structuring element". This element is related to the filter kernel utilized in linear spatial filters, with the constraint that it contains boolean values, i.e., either zero or one, or equivalently "true" or "false". Consequently, all locations assigned to "true" indicate that a pixel around the current center pixel (x, y) is part of the local neighborhood. For example, the neighborhood around a location (x, y) with the structure element

$$S_{\text{cross}} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{1.3.5.1}$$

contains the pixels $(x, y-1)$, $(x-1, y)$, (x, y), $(x+1, y)$, and $(x, y+1)$.

The presence of pixel within a neighborhood allows the definition of the "erosion" operation that assigns the current pixel $f(x, y)$ to the local minimum of the values in the neighborhood [51, Section 9.6.1, Page 666]:

$$[f \ominus b](x, y) = \min_{(s,t)\in b} \{f(x+s, y+t)\}. \tag{1.3.5.2}$$

If the image is a binary image, this is equivalent to only marking the current pixel as foreground, if all pixels within the neighborhood are foreground pixels [51, Section 9.2.1, Page 631-632]. Otherwise, the pixel will be re-classified as background.

The "dilation" function applies a local maximum, thus marking the pixel as foreground if any of the neighborhood pixels is classified as foreground [51, Section 9.2.2, Page 633]:

$$[f \oplus b](x, y) = \max_{(s,t)\in b} \{f(x+s, y+t)\}. \tag{1.3.5.3}$$

Dilation and erosion operations can be combined to form higher-order operators. Morphological "closing" is defined by the application a dilation followed by an erosion [51, Section 9.6.2, Page 668]:

$$[f \bullet b](x, y) = [(f \oplus b) \ominus b](x, y). \tag{1.3.5.4}$$

It yields an image where different structures within the neighborhood range are connected and holes smaller than the area are removed. A morphological "opening" applies an erosion followed by a dilation to remove objects smaller than the neighborhood area:

$$[f \circ b](x, y) = [(f \ominus b) \oplus b](x, y). \tag{1.3.5.5}$$

1.3.6 Classification

A generalization of segmenting pixels into a foreground and background class is to introduce different types of ROIs. For example, a pixel can be classified as background, or two distinct types of objects [51, Section 10.3.1, Page 739]. In the following text, we will use the term "classification" for the more generalized case of segmentation.

A naive classification method is the application of multiple intensity threshold levels that are associated with the types of foreground objects and the background class, respectively. Gonzalez and Woods present a three-way classification [51, Section 10.3.1, Page 739, Equation (10.3-2)], which we generalize into an operation that can be adapted to any number of classes $k > 1$. It assigns a class-specific value $c_1 \neq c_2 \neq \ldots \neq c_k$ to the each pixel based on whether the intensity is within the threshold limits given by $T_1, T_2, \ldots, T_{k-1}$:

$$g(x, y) = \begin{cases} c_1 & \text{if } f(x, y) \leq T_1 \\ c_2 & \text{if } T_1 \leq f(x, y) \leq T_2 \\ \ldots & \ldots \\ c_k & \text{if } T_1 \leq T_2 \leq \ldots < T_{k-1} \leq f(x, y). \end{cases} \tag{1.3.6.1}$$

As with the $k = 2$ case, a disadvantage is that the thresholds $T_1, T_2, \ldots, T_{k-1}$ need to be defined manually. The challenge is that it is not trivial to develop methods that generate the appropriate threshold values within a reasonable time. For example, Otsu's method is fast for two classes, but requires extensive improvements to avoid an exhaustive search over the exponential number of threshold combinations [83].

An alternative can be found in clustering methods that were designed to find groups within a set of points [87, Chapter 20, Page 284-285]. As images can be represented as set of points

$$\{(x, y, f(x, y)) | x \in \{1, \ldots, M\}, y \in \{1, \ldots, N\}\} \tag{1.3.6.2}$$

in a 3D space, these clustering techniques can also be applied for image classification.

A popular automated clustering technique is "K-means", which is based on the principle that points within the same cluster should have a small distance from each other. This distance measurement can be any metric [77], i.e., a function

$d(a, b)$ that satisfies the following constraints:

$$d(a, b) \geq 0, \tag{1.3.6.3}$$
$$a = b \Leftrightarrow d(a, b) = 0, \tag{1.3.6.4}$$
$$d(a, b) = d(b, a), \text{ and} \tag{1.3.6.5}$$
$$d(a, b) + d(b, c) \geq d(a, c). \tag{1.3.6.6}$$

The function should always return a positive number or zero if and only if the two points are equal. A distance metric must be symmetric, i.e., the function parameters can be switched without changing the results. Finally, $d(a, b)$ must adhere to the triangle inequality, meaning that a direct path between two points is never longer than a path through three points.

A central component of K-means is the definition of k "cluster centers" which are points that are used to partition the space into nearest neighborhood cells. Based on the location within the cluster space, each pixel is assigned to the closest cluster center, thus assigning it to a class. K-means starts with an initialization phase, where cluster centers are randomly assigned [87, Chapter 20.1, Page 286]. To find the final clusters, the algorithm repeats two steps until the solution converges: first, all points are assigned to the closest cluster center according to $d(a, b)$. The second step re-calculates the cluster centers based on the average of all assigned points, hence the name K-means. A trivial criterion to stop the algorithm is the case where the cluster centers are equal in two consecutive steps, reaching a fix point.

Clustering methods are part of a family of machine learning algorithms that are unsupervised, meaning that they take only the data to be grouped, estimated, or visualized into consideration [15, Chapter 1, Page 3]. A different approach is implemented by supervised learning that relies on labeled training data sets for their functionality.

1.3.7 Deep neural networks

An implementation of supervised learning is based around the functionality of neurons. The computer representation of such a cell is termed "perceptron" or "linear unit" and consists of $m > 1$ numeric inputs and a single numeric output, modelling the function of dendrites and axons, respectively [27, Chaper 1.1, Page 3-6] (see Figure 1.9).

A single perceptron is capable of applying classification tasks with two classes. To classify an image, a model is generated with all pixel values as inputs. For example, to classify an image with a size of 16×16 pixels, the perceptron is provided with 256 inputs, each containing the grayscale value of a different pixel. The output f_Φ is calculated by first weighting each input x_i by a factor w_i. The resulting values are then summarized and further modified by an additive bias parameter b. Based on the biased integrated weighted inputs, it is decided whether the output is zero or one, thus indicating the class of the whole image:

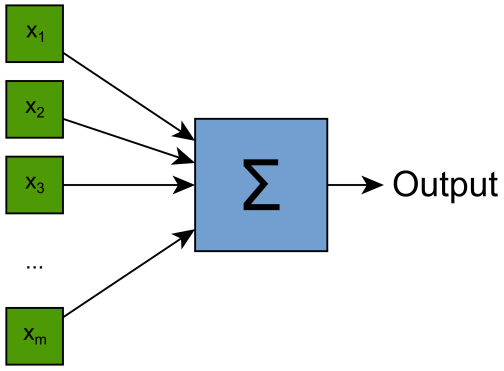


Figure 1.9: Computer model of a neuron. A perceptron or linear unit consists of m inputs (green) that are integrated into a single output (blue). Adapted from [27, Figure 1.3].

$$f_\Phi(x) = \begin{cases} 1 & \text{if } b + \sum_{i=1}^{m} x_i \cdot w_i > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{1.3.7.1}$$

The weight and bias parameters are not set manually by the user, but instead calculated automatically by the application of a training algorithm. The inputs of this operation are three independent and labeled data sets that are termed the "training data", "test data", and "validation data" [27, Chaper 1.1, Page 6-7]. The training data is the basis for calculating the weights and the bias, while the validation and test data sets are utilized to estimate the performance of the classifier. The difference between the validation and test sets is that the training algorithm incorporates only the measurements of the validation set to adjust the training parameters, while the test data set is only intended for external review. The training algorithm for a single perceptron starts with all parameters set to zero. Then, it iteratively improves the model by first comparing the results produced by the current settings with the expected classification and adjusting the parameters according to this difference.

The result is a trained binary classifier and performance metrics that can be obtained from the application of the test data set. As each perceptron produces a boolean result, a general classifier can be implemented by introducing multiple linear units into the model [27, Chaper 1.1, Page 8]. Each perceptron is then trained for detecting the presence of a specific class.

This shared "purpose" is the basis for the grouping of elements in a neural model into "layers" [27, Chaper 1.2, Page 9-11] (see Figure 1.10), specifically the "input layer" that contains the input data, and one or multiple "computational layers" that generate an output based on their inputs. The type of input is not restricted to external values from an input layer: by connecting the outputs of a computational layer to the inputs of a second computational layer, a deep neural network (DNN) is formed.

Compared to single-layered classification networks, perceptrons in a DNN have

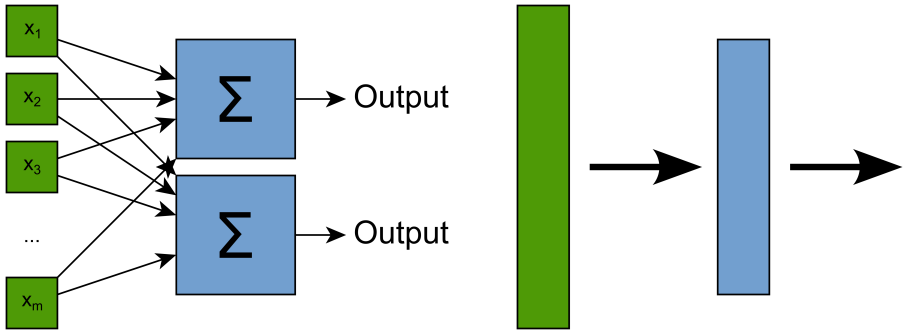


Figure 1.10: Representation of a neural model as set of layers. (a) Model with multiple perceptrons. **(b)** Representation of the model as input layer (green), computational layer (blue), and output layer (white). The arrows indicate that all elements of the previous layer are connected to all elements of the next layer. Adapted from [27, Figures 1.6-1.7].

two additional characteristics: non-linearity and a probabilistic output. The first property is introduced as additional non-linear "activation function" that is applied to the inputs of each perceptron prior to the multiplication with the weight [27, Chaper 2.3, Page 38-40]. The reason behind this additional operation is that a DNN with only linear units is mathematically equivalent to a single layer network and thus is subject to the same limitations. A common non-linear function applied in DNNs is the rectified linear unit (ReLU)

$$\rho(x) = \max(x, 0). \tag{1.3.7.2}$$

The probabilistic output is related to how the training of DNN is implemented. The already established method for training single-layer networks is not suitable for multi-layer networks. Instead, an alternative method is applied that for each layer i searches for better parameters Φ_i. This is achieved by minimizing a loss function $L(\Phi_i)$ that quantifies how poorly a network performs on the training data set [27, Chaper 1.2, Page 10]. The basis for this search is a gradient descent, which calculates the change in parameters by multiplying a learning rate $\mathcal{L}$ with the partial derivative of the loss function with respect to the parameter:

$$\Delta\Phi_i = -\mathcal{L}\frac{\partial L(\Phi)}{\partial \Phi_i}. \tag{1.3.7.3}$$

Consequently, the loss function must be differentiable, which is not the case if "steps" occur due to the comparison of discrete values. Thus, two modifications are made: first, each perceptron calculates a real value

$$f_\Phi(x) = b + \sum_{i=1}^{m} \rho(x_i) \cdot w_i\,; \tag{1.3.7.4}$$

second, all outputs of a layer are processed by a "softmax" function that divides each output by the sum of all output values in a layer [27, Chapter 1.2 and 2.3, Pages 11-12 and 39-40].

The training method based on gradient descent is also suitable for networks that are not fully connected, meaning that not all outputs of a perceptron are connected to all inputs in the next layer. A special case of such partially connected networks are convolutional neural networks (CNNs) that are characterized by the purpose of their parameters as weights of a linear spatial filter [27, Chaper 3.1, Page 53] (see section 1.3.2). By stacking multiple convolutional layers [27, Chaper 3.3, Page 61] and combining them with up- and down-scaling (pooling) [27, Chaper 3.4.3, Page 66] operations, CNNs are utilized in DNNs for image analysis.

A popular network based around CNNs is the "U-net" [113], which was designed to solve various image analysis tasks with the focus on minimizing the required training data and computational cost. The name is derived from its symmetric architecture that can be visualized into a form resembling the letter "U". The network first processes the data in a "contracting" arm consisting of multiple convolution layers, ReLU, and pooling operations. These produce images with a gradually lower resolution, thus putting an emphasis on features at various scales. This is followed by an "expansive" path that again applies convolutional and ReLU layers, but replaces the pooling with up-scaling operations. The outputs of the pooling operations are made available to the corresponding convolution stacks in the expansive paths.

The U-net and other architectures can be implemented manually, or via dedicated software frameworks designed for the application of deep learning, for example, *Tensorflow* [1] or *Pytorch* [106]. Based on these libraries, tools for the processing and analyzing of bioimage data sets were developed, including *StarDist* [121, 139], *CARE* [138], and *Cellpose* [126].

1.3.8 Image analysis tools

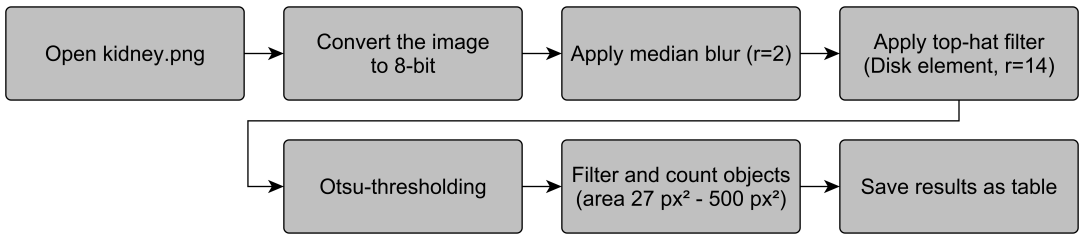
In sections 1.3.2 to 1.3.7, we briefly introduced methods that allow the processing and extraction of objects from images. To apply these operations, there are a variety of existing software tools that implement the image processing methods established in this thesis, as well as alternative and more advanced operations.

Many commercial and open source tools rely on a graphical user interface (GUI) to display images and provide users with options to apply analysis steps. Examples for commercial software include *Imaris* (Oxford Instruments) and *Arivis* (arivis AG), with both placing a focus on the aspect of rendering 3D data and interactive processing of images. For applying deconvolution with a PSF, there exist dedicated tools, for example *Huygens* (SVI), or *Imaris ClearView-GPU* (Oxford Instruments).

An alternative to commercial software are open source tools maintained by a community of developers and include *ImageJ* [115], *Icy* [35], and the *Microscopy image browser* [13]. These support the visualization of images on a computer screen and the execution of filtering steps with the purpose of forming a image processing pipeline. Especially *ImageJ* gained a wide popularity [122], due to the variety of functions that are flexibly integrated into the interface via a plugin system. Examples for such plugins include the deconvolution algorithm *DeconvolutionLab2* [116],

```
open("kidney.png");
selectWindow("kidney.png");
run("8-bit");
run("Median...", "radius=2");
run("Morphological Filters", "operation=[White Top Hat] element=Disk radius=14");
run("Auto Threshold", "method=Otsu white");
run("Analyze Particles...", "size=27-500 display clear summarize add");
saveAs("Results", "Summary.csv");
```

(a)



(b)

Figure 1.11: Visual representation of program code. **(a)** *ImageJ* macro code that opens an image with the name "kidney.png", applies a local median filter, a top-hat operation for background removal, Otsu-thresholding, and object size filtering. The results are saved as table. **(b)** Visual representation of (a) as flow chart. Nodes (gray boxes) correspond to the *ImageJ* operation. Arrows indicate the order of operations and flow of data between two nodes.

and the *TrackMate* [130] plugin that adds advanced methods for tracking biological objects in time series. As with commercial tools, there is open source software that specializes to specific image analysis tasks. *Cellprofiler* [91], for example, focuses on providing an intuitive workflow to set up and train machine learning models. *Cellpose* [126] was designed for the segmentation and resolution of cell clusters and other biological structures by the application of a DNN.

A GUI is not the only mode of interacting with a software. Various tools are restricted to a command line interface (CLI), or an application programming interface (API). Examples of such software libraries are *OpenCV* [20] and *Scikit Image* [137] that provide a standardized software interface to develop image analysis scripts. Many tools offer multiple interfaces. This is the case for *ImageJ* and *Icy* that each provide a GUI and a *Java* API that allows the development custom pipelines and interfaces. Additionally, these tools include interpreters for various script languages that enable the automated and reproducible execution of GUI functions. These can also be executed via a CLI to enable the analysis in a non-graphical environment.

A disadvantage of CLIs, APIs, and script languages is that they are only accessible to researchers with programming knowledge, which is hard to acquire due to the necessity of understanding program structures, solving of tasks via generation of computer procedures, and the adaption of the required syntax [107]. Additionally, researchers must learn strategies to resolve errors that require experience and a deep understanding of the inner workings of a computer.

An alternative to text-based scripts is based on the concept of representing programs as flow chart. Tools make use of this relationship to provide an entire visual mode of creating programs [75] (see Figure 1.11). These visual programming languages (VPLs) provide a GUI where users can freely arrange a predefined set of nodes. These encapsulate functions of the underlying API and allow the intuitive visualization of how data flows between their inputs and outputs. To execute a program formed from such a pipeline, a computer can for example, execute the node workloads in topological order, while ensuring that the required data transfer operations are applied. VPLs suitable for image analysis tasks include the protocol feature of *Icy* [35], image analysis plugins for the open source tool *KNIME* [14], and the "CLIJ Assistant" feature of *CLIJ* [55]. The concept is also featured in commercial tools, for example, *Apeer* (Carl Zeiss AG).

Due to the general applicability, VPLs are also utilized outside scientific data processing. Examples include *Godot* (`https://godotengine.org`), *Unity 3D* (Unity Software Inc.),*Davinci Resolve* (Blackmagic Design Pty. Ltd), and *Blender* (Blender Foundation).

1.4 Web applications and services

The software listed in the previous section is installed on the local hardware, thus requiring computers that have the necessary capabilities to execute the requested operations. The requirements depend on the software and for example, include a minimum in available random access memory (RAM), capabilities and speed of the central processing unit (CPU), and the presence of a graphics processing unit (GPU) with appropriate features and video random access memory (vRAM) capacity.

An alternative mode of software distribution is based on the ongoing developments in the world wide web that transitioned from static documents to a network of interconnected remote services [9]. An example for such a service in the field of bioimage analysis is the image data base *OMERO* [5] that provides standardized open protocols and related implementations for the remote storage of microscopy images and their metadata. Access to these machine-readable protocols is provided via clients that can be integrated into *ImageJ* or accessed from a web browser. This concept of a server that can be accessed via a web application has grown in popularity and is for example, implemented in *webKnossos* [18], and *ImJoy* [104].

There are multiple available protocols that facilitate the data transfer between the server and client. Due to the simplicity and efficiency, interfaces based on *Representational State Transfer* (REST) and *Javascript* object notation (JSON) have gained popularity and are already utilized in over 2000 websites [9]. The role of REST is to transfer messages over the hypertext transfer protocol (HTTP) via a set of basic operations for uploading, downloading, updating, and deleting remote resources. While REST makes no assumptions on the type of these resources, it is a common practice to send text in JSON format that is capable of encapsulating complex objects into a portable form that is independent of the utilized programming language. To implement the server or a non-web client, there are a

variety of existing libraries for many programming languages, including the *Spring* framework (VMware, Inc.), *Flask* [52], *Django* [37], or *ASP.NET* (Microsoft Corporation). The implementation of REST-based web clients is supported by many web application frameworks, including *React* (Meta Platforms, Inc.), and *Angular* (Google LLC).

Chapter 2

Objectives of this thesis

Characterizing the mechanisms behind the interactions between hosts and their pathogens is useful for the treatment of life-threatening conditions, including sepsis and hemolytic-uremic syndrome [11, 36]. Confocal microscopy, light sheet fluorescence microscopy (LSFM), multispectral optoacoustic tomography (MSOT), and other modern imaging modalities capture the interaction dynamics of multiple cells, or even whole organs. The resulting images allow the quantification of the visible processes, followed by the inference of their functionality via computer models. Due to the increasing volume of image data [111] and biological limitations on the achievable contrast [33], advanced computer programs must be developed that are capable of applying a rapid quantitative analysis without the need for manual intervention. With the ongoing adoption of digital data management, an important aspect of software has become its relationship with the principles of *Findability, Accessibility, Interoperability, and Reusability* (FAIR) [140]: first and foremost, computer programs must be FAIR, meaning that algorithms, and generated data should be open and reusable; second, tools should help to propagate the FAIR principles in the scientific community and especially interdisciplinary teams. These are essential to the image-based systems biology (IbSB) [93] approach that is based on a close collaboration between experimentalists and computer scientists. Here, software can help to remove existing barriers [90] and thus improve the adoption of new technologies and facilitate the generation of new knowledge.

The goal of this thesis is to contribute towards the propagation of the FAIR principles in the field of IbSB by the development of standardized, reproducible, high-performance, and accessible software for the quantification of interactions in biological systems.

In our first study, we focus on the extraction of pharmacokinetics via MSOT that are commonly quantified by an interactive approach. Here, an expert is required to manually determine the regions of interest (ROIs) that are the source for the statistical calculations, making the analysis of MSOT data slow and prone to researcher bias. Our study resolves following questions by the implementation of the *MSOT cluster analysis toolkit* (*Mcat*) that is based on *ImageJ*:

- Can the established manual approach successfully differentiate between healthy and septic mice?
- How to implement a fully automated, reproducible, and easy accessible computer software for the analysis of MSOT data?

Related to the utilization of *ImageJ* as platform to implement quantitative image pipelines was our second study. It is based around the scripting feature of *ImageJ* that is currently the common method to combine the features provided by this platform into purpose-built quantitative image analysis pipelines. As the requirement of programming knowledge introduces challenges in the collaboration between experimentalists and computer scientists, our study aimed to provide an alternative mode for creating pipelines via a visual programming language (VPL) called *Java image processing pipeline* (*JIPipe*) and answer the following questions:

- How to design a software that facilitates the collaboration between experimentalists and computer scientists?
- Can the functionality of *ImageJ* be encapsulated into a VPL?
- How to simplify the design of reproducible batch processing pipelines for the quantification of images?

The third study of this thesis, we aimed to utilize methods of microbiology and IbSB via *JIPipe* to further characterize the interactions between the soil-dwelling fungus *Mortierella verticillata*, its endosymbiotic bacteria, and the fungivorous nematode *Aphelenchus avenae*. The following questions were answered:

- Which toxin is produced by *Mortierella verticillata* to defend against *Aphelenchus avenae*?
- Are the endosymbiotic bacteria responsible for the production of the toxin?
- How can the effectiveness of toxin be quantified from a time series?

The final subject focuses on the development of high-performance image quantification pipelines that are necessary due to the big data volumes produced by various microscopy technologies, including LSFM. This study will address the following questions:

- Can the implementation of high-performance image quantification pipelines capable of processing big volumes of data be simplified?
- How to facilitate the access of high-performance tools to non-programmers?

Chapter 3

Overview of manuscripts

Spatial quantification of clinical biomarker pharmacokinetics through deep learning-based segmentation and signal-oriented analysis of MSOT data

Authors Bianca Hoffmann, Ruman Gerst, Zoltán Cseresnyés, Wan-Ling Foo, Oliver Sommerfeld, Adrian T. Press, Michael Bauer, Marc Thilo Figge

Status Published in *Photoacoustics* Volume 26, June 2022, 100361

Summary In this publication we introduce our new machine learning-based approach for the quantification of pharmacokinetics via multispectral optoacoustic tomography (MSOT). We showed that our software was capable of differentiating healthy from septic mice by tracking indocyanine green (ICG) dynamics, while the standard approach was generally unsuccessful. To make our algorithm accessible to non-programmers, we developed a plugin for the widely used *ImageJ* platform.

Author's contribution ZC, OS, ATP, MB and MTF conceptualized the study. WF, OS and ATP designed and performed animal experiments. BH, RG, ZC and MTF developed image analysis methodology. BH and RG developed the presented software. BH performed the quantitative image analysis. BH, RG and ZC validated the image analysis methodology and analyzed the data. BH and RG wrote the original draft and prepared figures. ATP, MB and MTF supervised the study. MB and MTF did project administration and funding acquisition. All authors revised the manuscript and approved it for submission.

JIPipe: Visual batch processing for ImageJ

Authors Ruman Gerst, Zoltán Cseresnyés, Marc Thilo Figge

Status Submitted to *Nature Methods* (Preprint published to *Research Square*)

Summary This publication introduces a new software that simplifies the design of quantitative image analysis pipelines with *ImageJ* by encapsulating its functionality into a visual programming language (VPL) termed *Java image processing pipeline* (*JIPipe*), thus combining the reproducibility of programming languages with the accessibility of flow charts. Due to the standardized data and algorithm model and its graphical user interface (GUI), our tool facilitates both the implementation and propagation of *Findability, Accessibility, Interoperability, and Reusability* (FAIR).

Author's contribution ZC and MTF conceptualized the study. RG developed and tested the software. ZC tested the software and designed the example workflows. RG and ZC wrote the initial manuscript, which was critically revised by MTF.

Bacterial endosymbionts protect beneficial soil fungus from nematode attack

Authors	Hannah Büttner, Sarah P. Niehs, Koen Vandelannoote, Zoltán Cseresnyés, Benjamin Dose, Ingrid Richter, Ruman Gerst, Marc Thilo Figge, Timothy P. Stinear, Sacha J. Pidot, and Christian Hertweck
Status	Published in *PNAS* September 14, 2021 118 (37)
Summary	This publication investigates the interaction between the soil-dwelling fungus *Mortierella verticillata*, and the mycophagous nematode *Aphelenchus avenae*. By a combination of wet lab methods and our advanced quantitative image analysis pipeline based on *JIPipe*, we could show that the fungus is protected by cytotoxic metabolites produced by its bacterial endosymbiont.
Author's contribution	HB, SPN, and CH designed research. HB, SPN, KV, ZC, BD, IR, RG, and SJP performed research. HB, SPN, KV, ZC, BD, IR, RG, MTF, TPS, and SJP analyzed data; and HB, SPN, SJP, and CH wrote the paper.

MISA++: A standardized interface for automated bioimage analysis

Authors Ruman Gerst, Anna Medyukhina, Marc Thilo Figge

Status Published in *SoftwareX* 11 (2020): 100405.

Summary In this publication we present a framework that simplifies the development of high-performance image analysis tools with the machine-oriented programming language *C++*. The software provides highly standardized components for exchanging parameters and storage of data and metadata, thus facilitating the implementation of the FAIR guidelines. To improve the accessibility of high-performance quantification pipelines to non-programmers, we developed a standardized GUI built on *ImageJ* that automatically adapts to any software developed with our framework.

Author's contribution AM and MTF conceived and designed the study. RG developed the C++ framework and high-performance kidney analysis. AM provided the original Python implementation and contributed the data, metadata, and explanations. RG wrote the initial manuscript, which was critically revised by AM and MTF.

Chapter 4

Manuscripts

4.1 Spatial quantification of clinical biomarker pharmacokinetics through deep learning-based segmentation and signal-oriented analysis of MSOT data

Manuskript Nr. 1

Titel des Manuskriptes: Spatial quantification of clinical biomarker pharmacokinetics through deep learning-based segmentation and signal-oriented analysis of MSOT data

Autoren: Bianca Hoffmann, Ruman Gerst, Zoltán Cseresnyés, WanLing Foo, Oliver Sommerfeld, Adrian T. Press, Michael Bauer, Marc Thilo Figge

Bibliographische Informationen: Hoffmann, B., Gerst, R., Cseresnyés, Z., Foo, W., Sommerfeld, O., Press, A. T., ... & Figge, M. T. (2022). Spatial quantification of clinical biomarker pharmacokinetics through deep learning-based segmentation and signal-oriented analysis of MSOT data. Photoacoustics, 100361.

Der Kandidat / Die Kandidatin ist (bitte ankreuzen)

☐ Erstautor/-in, ☒ Ko-Erstautor/-in, ☐ Korresp. Autor/-in, ☐ Koautor/-in.

Status: Veröffentlicht in Photoacoustics

Anteile (in %) der Autoren / der Autorinnen an den vorgegebenen Kategorien der Publikation

Autor/-in	Konzeptionell	Datenanalyse	Experimentell	Verfassen des Manuskriptes	Bereitstellung von Material
Bianca Hoffmann	30 %	25 %	25 %	30 %	0 %
Ruman Gerst	30 %	30 %	20 %	30 %	0 %
Zoltán Cseresnyés	10 %	20 %	25 %	10 %	0 %
Marc Thilo Figge	10 %	15 %	0 %	10 %	50 %
Weitere	20 %	10 %	30 %	20 %	50 %
Summe:	100 %	100 %	100 %	100 %	100 %

_____ _____

Unterschrift Kandidat/-in Unterschrift Betreuer/-in (Mitglied der Fakultät)

Photoacoustics 26 (2022) 100361

Contents lists available at ScienceDirect

Photoacoustics

journal homepage: www.elsevier.com/locate/pacs

Spatial quantification of clinical biomarker pharmacokinetics through deep learning-based segmentation and signal-oriented analysis of MSOT data

Bianca Hoffmann [a,1], Ruman Gerst [a,b,1], Zoltán Cseresnyés [a], WanLing Foo [c,d], Oliver Sommerfeld [c,d], Adrian T. Press [c,d,e], Michael Bauer [c,d], Marc Thilo Figge [a,d,f,*]

[a] Research Group Applied Systems Biology, Leibniz Institute for Natural Product Research and Infection Biology - Hans Knöll Institute (HKI), Beutenbergstr. 11a, 07745 Jena, Germany
[b] Faculty of Biological Sciences, Friedrich Schiller University Jena, Bachstr. 18k, 07743 Jena, Germany
[c] Department of Anesthesiology and Intensive Care Medicine, Jena University Hospital, Am Klinikum 1, 07747 Jena, Germany
[d] Center for Sepsis Control and Care, Jena University Hospital, Am Klinikum 1, 07747 Jena, Germany
[e] Medical Faculty, Friedrich Schiller University Jena, Kastanienstr. 1, 07747 Jena, Germany
[f] Institute of Microbiology, Faculty of Biological Sciences, Friedrich Schiller University Jena, Neugasse 25, 07743 Jena, Germany

ARTICLE INFO

Keywords:
Multispectral optoacoustic tomography
Quantitative image analysis
Deep learning
ImageJ plugin
Biomarkers
Pharmacokinetics
Sepsis

ABSTRACT

Although multispectral optoacoustic tomography (MSOT) significantly evolved over the last several years, there is a lack of quantitative methods for analysing this type of image data. Current analytical methods characterise the MSOT signal in manually defined regions of interest outlining selected tissue areas. These methods demand expert knowledge of the sample anatomy, are time consuming, highly subjective and prone to user bias. Here we present our fully automated open-source MSOT cluster analysis toolkit *Mcat* that was designed to overcome these shortcomings. It employs a deep learning-based approach for initial image segmentation followed by unsupervised machine learning to identify regions of similar signal kinetics. It provides an objective and automated approach to quantify the pharmacokinetics and extract the biodistribution of biomarkers from MSOT data. We exemplify our generally applicable analysis method by quantifying liver function in a preclinical sepsis model whilst highlighting the advantages of our new approach compared to the severe limitations of existing analysis procedures.

1. Introduction

Multispectral optoacoustic tomography (MSOT) has become increasingly recognised as a non-invasive imaging modality for pre-clinical and clinical research [1]. It serves as a powerful diagnostic tool for a large variety of diseases, including breast cancer [2], melanoma [3], vascular disorders [4] and Crohn's disease [5]. In addition to the capability to conduct in vivo studies, MSOT offers the possibility to collect time-resolved, functional tissue information at high spatio-temporal resolution for multiple photoabsorbing molecules at low costs [6].

However, quantitative analysis of MSOT image data is currently based on the manual assignment of regions of interest (ROI) within specific anatomical areas from which basic signal intensity statistics are extracted and followed over time [7–13]. We will refer to this procedure by the term *tissue-oriented* analysis because it quantifies signal change over time within a specifically chosen tissue region. While this approach is intuitive and allows a seemingly straightforward interpretation of the measurements, it suffers from several weaknesses: (i) manual definition of ROIs can be time consuming and demands expert knowledge of the anatomy and physiology of the sample, (ii) ROI definition is prone to user bias, (iii) any bias may reduce the reproducibility of study results and (iv) hinder the comparability of results across studies. Furthermore, since the attenuation of light in deep tissue limits the penetration depth

Abbreviations: AUC, Area under the curve; DAG, Directed acyclic graph; DL, Deep learning; GUI, Graphical user interface; ICG, Indocyanine green; Mcat, MSOT cluster analysis toolkit; MSE, Mean squared error; MSOT, Multispectral optoacoustic tomography; ROI, Region of interest; PCI, Peritoneal contamination and infection; WAC, Weighted-average curve.

* Corresponding author at: Research Group Applied Systems Biology, Leibniz Institute for Natural Product Research and Infection Biology - Hans Knöll Institute (HKI), Beutenbergstr. 11a, 07745 Jena, Germany.
E-mail address: thilo.figge@leibniz-hki.de (M.T. Figge).

[1] Authors contributed equally.

https://doi.org/10.1016/j.pacs.2022.100361
Received 13 February 2022; Received in revised form 7 April 2022; Accepted 22 April 2022
Available online 26 April 2022

of MSOT, even homogeneous tissue structures can show highly different signal intensities at different depths [14]. Consequently, differences between experimental conditions may be entirely obscured by these uncertainties of the analysis procedure and prevent utilising MSOT where most sensitive and accurate quantification is desired, e.g. when investigating potential new drugs or aiming for predictions in personalised medicine.

In contrast to the above-mentioned tissue-oriented analysis, we introduce a *signal-oriented* approach, which does not rely on the exact definition of a specific tissue region and thereby enhances the objectiveness of the analysis. Pixel-wise clustering of the observed signal kinetics is performed including the entire investigated sample, e.g. a whole mouse cross-section, to extract the pharmacokinetics and biodistribution of a particular photoabsorber. We compare the results of our method to the current standard approach of tissue-oriented analysis. Furthermore, we illustrate how the utilisation of confined ROIs can alter the outcome of quantitative MSOT image analysis, likely leading to incorrect results and erroneous conclusions. We developed the MSOT cluster analysis toolkit *Mcat*, which is provided as a plugin for the widely used open-source image analysis software *ImageJ* [15] and can be applied by users with no professional expertise in image processing. It is freely available at https://github.com/applied-systems-biology/mcat (https://doi.org/10.5281/zenodo.6046031).

We exemplify *Mcat* by applying it to MSOT data of mice with a life-threatening polymicrobial infection accompanied by liver dysfunction, i.e. peritoneal contamination and infection model (PCI) of sepsis [16, 17]. Liver function is quantitatively assessed by MSOT imaging of the clinically approved fluorescent chromophore indocyanine green (ICG) [18]. Hepatic ICG clearance is compared between healthy animals and those subjected to PCI. Since ICG is almost exclusively eliminated by the liver when administered intravenously, its excretion is impaired by liver dysfunction under septic condition and it therefore serves as a marker for liver function [18]. Our fully automated signal-oriented analysis outperforms the tissue-oriented approach in distinguishing healthy from diseased animals and provides information about the spatial distribution of the biomarker. Furthermore, the presented method is not limited to any particular photoabsorber or anatomical region; thus, the main steps of this analysis can be applied to virtually any type of time-resolved image data.

2. Methods

This section will first describe the segmentation of MSOT images, their pre-processing and then introduce our newly developed signal-oriented analysis approach. Subsequently, we describe the tissue-oriented approach, which is currently the standard procedure for the quantitative analysis of MSOT image data, followed by an overview of utilised software and the statistical analysis. Finally, the experimental methods are explained that were used to validate our new signal-oriented analysis.

2.1. Image segmentation

For automated segmentation of the animal cross-sections, we utilised a deep learning (DL) approach based on the software *Cellpose*, an algorithm designed to identify biological objects that is based on a U-net architecture with residual blocks [19]. The complete workflow is outlined in Fig. 1.

The training dataset was provided by three experimentalists who drew ROIs around animal outlines for five random and unique time frames per animal (see Supplementary Fig. A1 for more details on data selection). The manual annotation and subsequent training of the *Cellpose* neural network model were performed with the visual programming language *JIPipe* [20]. The final dataset consisted of 315 images from which 80% were used for training, 10% for validation, and 10% for testing of the model. We configured *Cellpose* to utilise no pre-trained model, run for 1000 epochs and assume the object diameter to be 210 pixels. The default settings were kept for all other parameters: loss function = L2 and cross-entropy, learning rate = 0.2, batch size = 0.8, augmentation = [random rotation (0–360°), random scaling (0.5x to 1.5x), random translations].

The trained model was then applied to the first time frame (t=0) of each of the image stacks. The resulting probability maps were segmented in *Mcat* with *ImageJ* functions. The probability maps were first convolved with a median filter of radius 3, the contrast was enhanced with saturation set to 0.3 and then binarised with Otsu thresholding. The remaining holes were filled, noise was removed, and only the largest foreground object was kept. Next, morphological closing and opening with radii of 50 pixels from the plugin *MorphoLibJ* [21] were applied to smoothen the ROI outline. The canvas size was doubled before this step to avoid artefacts at the image borders and afterwards reduced to the original image size. Finally, the convex hull of the foreground object was created and used as the final ROI, which will be from now on referred to as animal-ROI.

2.2. Image pre-processing

The multidimensional MSOT image stacks (see Methods Section 2.6.2 for details on image acquisition) were pre-processed prior to image analysis. The water and ICG image channels were extracted, and rigid

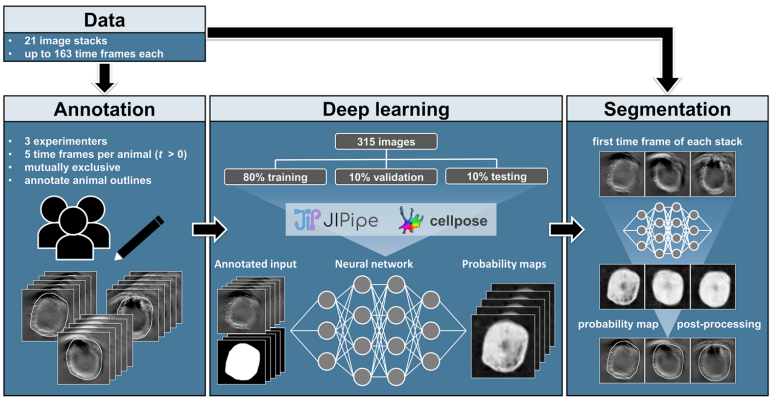


Fig. 1. DL-based segmentation of MSOT images. For the manual annotation, three experimenters drew animal outlines in five random and unique time frames ($t > 0$) for each of the 21 animals. Based on these 315 annotated images, a *Cellpose* neural network was trained from scratch using the visual programming language *JIPipe* [20], where 80% of the images were used for training and 10% each for validation and testing. The final segmentation of the first time frame ($t = 0$) of each MSOT image stack was then performed with the trained neural network. The resulting probability maps were post-processed with basic image processing functions to obtain the final ROIs outlining the animals.

registration was performed on the water channel. The resulting transformation matrix was then applied to the ICG channel to reduce motion artefacts. To compensate for signal intensity differences between MSOT scans, the ICG channel was z-transformed for each animal according to

$$Z(x,y,z) = \frac{I(x,y,z) - \mu_{t0}}{\sigma_{t0}}, \quad (1)$$

where $Z(x,y,z)$ is the z-transformed pixel value at position (x,y,z), $I(x,y,z)$ is the original pixel intensity value at position (x,y,z), μ_{t0} is the mean signal intensity value of the first time frame of the ICG channel, and σ_{t0} is the corresponding standard deviation. The image stacks were then smoothed by downsampling in the z-direction to compensate for animal breathing. This factor is denoted as smoothing factor and was set to $s = 4$, i.e. the signal intensity values of four subsequent time frames were averaged into one new time frame. Finally, the pixel intensity values were transformed into time derivatives of pixel intensities between consecutive time frames. Thus, the analysis was performed on signal intensity change over time, to avoid dealing with negative signal intensity values that may be introduced by image reconstruction and spectral unmixing.

2.3. Signal-oriented analysis

The details on our newly developed signal-oriented analysis approach are presented along with the results in Section 3.2. In brief, it automatically identifies connected regions of similar signal kinetics from the animal-ROIs that were obtained by automated image segmentation. We therefore utilised k-means clustering, where the parameter k defines the number of clusters with different signal kinetics. All animals from one group were mixed together to compare treatment groups, and clustering was performed on these pooled datasets. The overall extent of biomarker signal increase over time was calculated for each animal based on the kinetic clusters reflecting a signal net increase and their corresponding pixel frequencies. Additionally, colour-coded images of the spatial distribution of all kinetic clusters were reconstructed for each animal to visualise the biodistribution of ICG.

2.4. Tissue-oriented analysis

For the tissue-oriented analysis, an expert labelled the liver tissue in the water channel, referred to as liver-ROIs (see Results Section 3.3). We then extracted the mean, maximum and 95th-percentile intensity values within the liver-ROIs from the ICG channel of the pre-processed images. These signal statistics were calculated for each time frame independently, yielding time curves for each animal. Since the number of time frames per MSOT image stack was not always identical, excess time frames were excluded from the analysis. We then calculated the area under the curve (AUC) for the time curves of the three signal statistics to compare treatment groups.

To investigate the sensitivity of ROI placement in MSOT images, we placed a circular ROI with a diameter of ten pixels in the outer region of liver tissue. The ROI was then shifted four times in the y-direction in 5-pixel steps and three times in the x- and y-direction in 5-pixel steps, respectively, to obtain multiple samples of liver tissue within close proximity (5 pixels ≈ 0.38 mm; see Results Section 3.3). As before, the mean, maximum and 95th-percentile signal intensity values were extracted from each of these ROIs and the liver-ROI, and they were followed over time to assess the homogeneity of the ICG signal within the liver.

2.5. Utilised software and statistical analysis

The signal-oriented analysis, including the pre-processing of MSOT images, was implemented in *Java* (see Supplementary Table A2 for details on utilised software and libraries). The tissue-oriented analysis and the investigation of its sensitivity towards ROI placement were performed with the open-source image analysis software *Fiji* (ImageJ v1.52p) [15]. Statistical analysis was performed using the statistics software *R* v3.6.1. Differences between groups were evaluated using a two-tailed Wilcoxon rank sum test. Effect sizes were calculated as Hedges' g using the *cohen.d* function of the *R* package *effsize* (parameters: *pooled* = *TRUE*, *hedges.correction* = *TRUE*, *paired* = *FALSE*), where g is interpreted as indicative for a low effect for values around $g = 0.2$, as a medium effect for values around $g = 0.5$ and as a large effect for values around $g = 0.8$ and larger. All p-values and effect sizes were calculated for $n_{Sham} = 9$ and $n_{PCI} = 12$ animals. Boxplot centre lines denote median values. Mean values are represented as black triangles.

We screened a range of values for the two main parameters smoothing factor s and k-means k to perform parameter estimation of the signal-oriented analysis. The whole analysis was repeated five times for each parameter combination to also consider the random initialisation step of k-means clustering. The robustness of a specific parameter combination was evaluated by calculating the weighted mean squared error (MSE) of the mean effect size ($\bar{g}$) between treatment groups of the five runs with regard to $\bar{g}$ of the neighbouring parameter combinations (see Results Section 3.4). The MSE of parameter combination (s,k) was calculated as

$$MSE_{s,k} = \frac{1}{|N4(s,k)| + |N8(s,k)|}\left(\sum_{(s',k')\in N4(s,k)} \left(\bar{g}_{s,k} - \bar{g}_{s',k'}\right)^2 + \sum_{s',k'\in N8(s,k)} 0.5\ \left(\bar{g}_{s,k} - \bar{g}_{s',k'}\right)^2\right), \quad (2)$$

where $N4(s,k)$ are all 4-connected nearest-neighbour parameter combinations of (s,k), $N8(s,k)$ are all 8-connected nearest-neighbour parameter combinations of (s,k) that are not included in $N4(s,k)$ and $\bar{g}_{s,k}$ is the mean effect size of parameter combination (s,k). MSE values were not calculated for parameter combinations at the borders of the parameter space because the numbers of connected neighbours are reduced for these combinations. To find the best parameter combination in terms of $\bar{g}$ and MSE, we scaled both values to the same range [0,1] and subtracted the resulting MSE values from 1. The scaled values of $\bar{g}$ were calculated as

$$\bar{g}(s,k)_{scaled} = \frac{\bar{g}_{s,k} - \min(\bar{g})}{\max(\bar{g}) - \min(\bar{g})}, \quad (3)$$

where $\bar{g}(s,k)_{scaled}$ is the scaled value of $\bar{g}_{s,k}$, $\bar{g}_{s,k}$ is the mean effect size for the parameter combination (s,k), and $\min(\bar{g})$ and $\max(\bar{g})$ are the minimum and maximum values of all $\bar{g}_{s,k}$. Similarly, the scaled MSE values were calculated as

$$MSE(s,k)_{scaled} = 1 - \frac{MSE_{s,k} - \min(MSE)}{\max(MSE) - \min(MSE)}, \quad (4)$$

where $MSE(s,k)_{scaled}$ is the scaled value of $MSE_{s,k}$, $MSE_{s,k}$ is the MSE for the parameter combination (s,k), and $\min(MSE)$ and $\max(MSE)$ are the minimum and maximum values of all $MSE_{s,k}$.

Descriptive statistics for all group comparisons can be found in the Supplementary Information A3.

2.6. Experimental confirmation

To confirm the advantage of our signal-oriented approach over the tissue-oriented analysis, we applied both concepts to an established preclinical sepsis model and quantitatively evaluated the liver function of healthy and diseased animals.

2.6.1. Sepsis animal model

All experimental protocols involving animals were approved by the State Administration Office of Thuringia, Germany. Eight to twelve weeks old male and female FVB/NRj inbred mice were housed under specific pathogen-free conditions in the animal facility of the Jena University Hospital under an artificial 12 h/12 h dark/light cycle with 20 min dim phases at 21 ± 2 °C and humidity of 55% ± 10%. Animals had access to standard rodent chow and drinking water ad libitum. Food soaked in water was additionally offered on the ground during infection. A life-threatening infection accompanied by an early liver failure, i.e. sepsis, was induced in 12 animals by the peritoneal contamination and infection model (PCI group). Therefore, approximately 60 µl of a characterised human stool suspension were injected per animal [16]. A control group of 9 animals (Sham group) received an intraperitoneal injection with an equal volume of a sterile salt solution (Ringer acetate, Fresenius Kabi). Approximately 2.5 mg metamizole dissolved in drinking water was given orally as drops on the animals' tongue at infection and then every 6 h for pain relief. Further, both groups received either the broad-spectrum antibiotic meropenem subcutaneously 6 h and 18 h after PCI or a salt solution treatment (2.5 mg^{-1} kg body weight dissolved in Ringer acetate to a volume of 2.5 mg mL^{-1}). The liver function was evaluated by MSOT imaging 24 h post infection.

2.6.2. Multispectral optoacoustic tomography

MSOT imaging was performed 24 h after either PCI induction or injection of sterile salt solution using the inVision 256-TF MSOT system (iTheraMedical, Munich, Germany). Animals were anaesthetised throughout MSOT preparation and image acquisition with 1.5–2% of isoflurane vaporised in oxygen. The abdominal imaging area was shaved using an electric shaver and commercial hair removal cream. Animals were positioned in the imaging chamber to scan a cross-section of the liver. Images were acquired at six wavelengths (715 nm, 730 nm, 760 nm, 800 nm, 850 nm and 900 nm) per time frame. Ten images were averaged for each wavelength and time frame to reduce animal motion and breathing artefacts. Approximately 20 µg ICG (Verdye, Diagnostic Green, USA) were administered intravenously through a tail-vein catheter 2 min after image acquisition was started. Imaging lasted for 20 min in total. Raw MSOT images were reconstructed by model-based backprojection (filter range: 50 kHz to 6.5 MHz) and spectrally unmixed into four channels (water, ICG, deoxygenated haemoglobin, oxygenated haemoglobin) by linear regression using the proprietary software ViewMSOT v3.8.1.04 (iTheraMedical, Munich, Germany).

3. Results

This section will first address the accuracy of automated image segmentation, followed by a detailed description of our new signal-oriented analysis of MSOT images and its performance in quantification of liver function in comparison with the standard approach. Subsequently, we will highlight the robustness of our new approach regarding parameter choice and segmentation accuracy and finally introduce the implementation as a readily available software toolkit.

3.1. Deep learning-based approach enables accurate automated segmentation of MSOT image data

The DL-based segmentation extracted the animal outline for each of the 21 MSOT image stacks. The identified animal-ROIs were visually very close to the manual annotations of each of the three experimenters, as exemplified for three representative animals in Fig. 2a. The pairwise Dice coefficient, which measures the similarity of two annotations in terms of overlap, shows a high concordance between the manual annotations and the automated segmentation (Fig. 2b). All pairwise comparisons reached high Dice coefficients ranging from 0.963 to 0.98, indicating an overlap of over 95% for each of the annotations. The lowest agreement was found between experimenters two and three and the highest agreement between experimenters one and three. The agreement between all the manual annotations and the DL-based segmentation reached intermediate values ranging from 0.969 to 0.977.

3.2. Signal-oriented analysis determines biomarker pharmacokinetics and biodistribution

Our new signal-oriented approach automatically identifies connected regions within the segmented area that exhibit similar signal kinetics of the analysed biomarker (Fig. 3).

All pixels outside the animal-ROIs determined by DL-based segmentation are set to zero, and the image stacks are cropped to the bounding boxes of the animal-ROIs. These steps reduce the impact of potentially noisy background regions, which can otherwise also comprise a varying proportion of the image area depending on the animal size, during the following analysis. Each pixel position in the preprocessed ICG image stacks holds a vector of time derivative values describing the ICG signal change over time for this spatial position. These vectors of length n serve as n-dimensional feature vectors for k-means clustering to obtain clusters reflecting the most prominent signal kinetics. The parameter k is used to control how many of these kinetic clusters are derived. Alongside, colour-coded images of their spatial distribution are reconstructed to visualise the biomarkers' distribution. To this end, a new two-dimensional image is created for each animal where each pixel value is set to a predefined colour, depending on which kinetic cluster has the smallest Euclidean distance to the time derivative vector of this pixel position. To compare treatment groups, all vectors

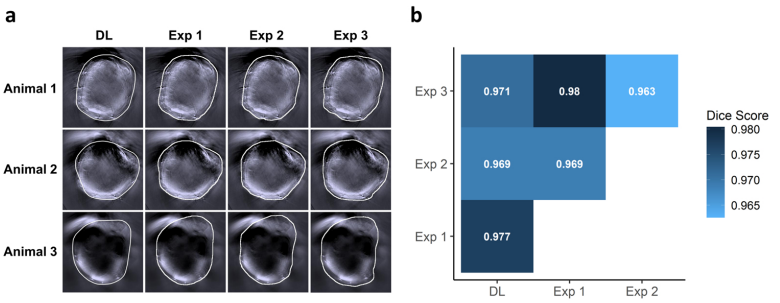


Fig. 2. Accuracy of DL-based segmentation in comparison to manual annotation. (a) The manual annotations of three experimenters and the DL-based segmentation are shown for three representative animals as white overlays on the first time frame of each of the image stacks. Besides minor variations, all outlines are visually very close and there is no apparent difference between the automated and manual segmentations. (b) The pairwise Dice coefficients show a high concordance between all the segmentations, where the lowest agreement was observed between two of the three experimenters (n = 21 for each pairwise comparison).

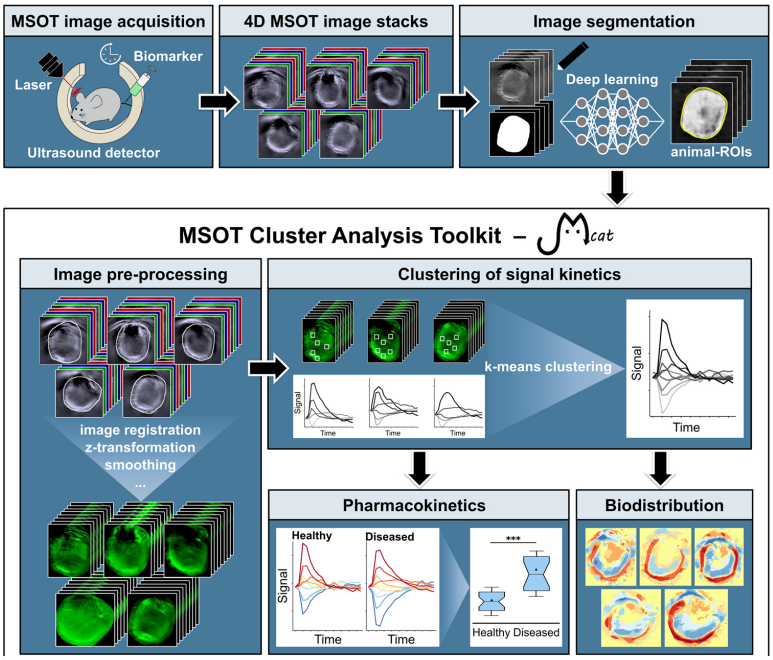


Fig. 3. Scheme of new signal-oriented analysis. The biomarker ICG was injected shortly after the start of the MSOT image acquisition, and the collected time-resolved MSOT image stacks were spectrally unmixed into four channels (water, ICG, deoxygenated haemoglobin and oxygenated haemoglobin). ROIs were derived with a DL-based approach and are denoted as animal-ROIs (yellow outline). The image stacks were then pre-processed, and the primary signal kinetics present in the ICG channel were extracted by k-means clustering on the pixel level. The obtained pharmacokinetics were then used to compare treatment groups quantitatively. Additionally, the biodistribution of the extracted signal kinetics were reconstructed and visualised per animal.

from all animals in one treatment group, i.e. Sham or PCI, are mixed and clustering is performed on these combined vector sets. The clustering is executed 100 times, and the result with the lowest overall Euclidean distance of the kinetic clusters to the corresponding data points is picked for subsequent analysis, accounting for the random initialisation step of the k-means algorithm.

Since we aim to analyse ICG uptake and clearance through the liver, we focus on kinetic clusters that reflect a signal net increase over time. This is assumed to be the case if the cumulative value of the curve of a kinetic cluster is larger than ε. The definition of this offset is necessary to ensure that only curves with a signal increase are considered and that curves with only slight deviations from the zero line are excluded as these are associated with regions where the ICG signal is not substantially changing over time. We set $\varepsilon = 0.1$ based on the observed typical variations around the zero line. For each animal, we calculate a normalised weighted-average-curve (WAC) according to

$$WAC_i = \frac{\sum_{j=1}^{n} C_j \bullet F_{i,j}}{x_i \bullet y_i}, \tag{5}$$

where WAC_i is the weighted-average-curve of animal i, C_j is the curve of the j-th kinetic cluster with a signal net increase, $F_{i,j}$ is the frequency of C_j for animal i, x_i is the width of the image stack of animal i, and y_i is the corresponding height. AUC values are then calculated based on the WACs to compare treatment groups.

3.3. New signal-oriented analysis outperforms current standard approach

Mice were treated either with PCI or a sterile salt solution 24 h before MSOT imaging to compare the liver function between diseased and healthy animals. No animal died or reached an experimental endpoint, e.g. lethargy, within this 24 h period. In addition, all PCI animals exhibited clinical signs of infection, including reduced activity and reactivity to external stimuli, unconditioned fur and diarrhoea, while the appearance and behaviour of animals in the Sham group were within normal limits.

With our new signal-oriented analysis, we extracted ten kinetic clusters for both Sham and PCI treatment groups, capturing the most prominent signal kinetics present in the ICG channels of the respective treatment group (Fig. 4a; see Results Section 3.4 for information on parameter choice). For Sham animals, one more of the ten kinetic clusters reflected a signal increase compared to PCI animals. Additionally, one kinetic cluster of the PCI group exhibited a constant signal increase over time (cluster 8), which could not be observed in the Sham group.

The curves of the kinetic clusters and corresponding pixel frequencies were used to calculate a WAC for each animal (Fig. 4b), and the difference between Sham and PCI treatments was quantified by calculating the AUC values of the WACs (Fig. 4c). PCI animals had significantly higher AUC values than Sham animals, which was also reflected by a very large effect size between the two groups ($p = 1.3 \bullet 10^{-4}$, $g = 1.83$).

The WACs showed a substantial initial increase of ICG signal for both Sham and PCI animals. This increase peaked at around 4 min after the acquisition was started and slowed down subsequently. We note that ICG signals are always reported as time derivatives and not as absolute signal intensity values. For some animals, signal change turned into a signal decrease, which is expected as a result of the ICG clearance by the liver.

In this phase, PCI animals exhibited constantly higher values of ICG signal than Sham animals, which led to a higher overall ICG signal intensity detected in the liver tissue. The plateau phase of the ICG signal was reached later for PCI animals: starting at around 9 min for Sham and 13 min for PCI (see dashed line in Fig. 4b). The ICG signal decreased at this time point for healthy animals, whereas it continued to rise for seven out of the twelve PCI mice. Generally, these animals were characterised by lower peak signal intensity, albeit higher signal increase in the second

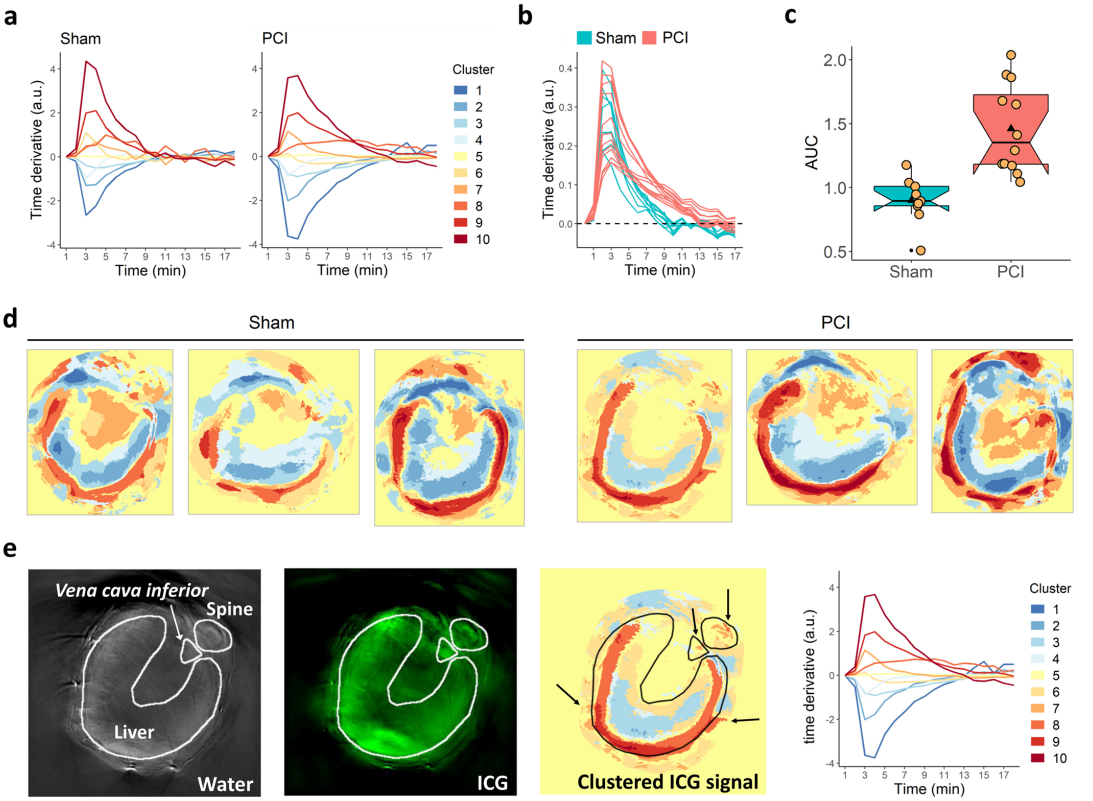


Fig. 4. Results from the signal-oriented analysis. (a) Main ICG signal kinetics were obtained by k-means clustering with $k = 10$ for Sham and PCI animal groups. (b) For each animal, a WAC of all curves of kinetic clusters with a net increase, i.e. curves reflecting ICG uptake, was calculated with the weights being the pixel frequencies of the respective kinetic clusters for this animal. The dashed line marks the change from a signal increase (positive values) to signal decrease (negative values). (c) AUC values were calculated for the WACs and are visualised as boxplots for Sham and PCI groups, with the black triangles denoting mean values and the yellow dots being individual data points (two-tailed Wilcoxon rank sum test $p = 1.3 \bullet 10^{-4}$, Hedges' $g = 1.83$). (d) Colour-coded images reveal the heterogeneous distribution of ICG signal across liver tissue (three representative images are depicted for each treatment group). (e) ROIs were drawn around liver tissue, spine and *Vena cava inferior* in the water channel by an expert and were overlaid on the ICG channel and the colour-coded visualisation of ICG biodistribution. The right panel depicts the corresponding kinetic curves. The liver tissue was strongly heterogeneous with respect to ICG kinetics, and smaller clusters of ICG signal increase fitted to the positions of the spine, the *Vena cava inferior* and vessels (arrows).

half of image acquisition than other animals from this group. This difference became even more apparent when looking at the time curves of integrated signal intensity (see Supplementary Fig. A4).

Representative images of the spatial distribution of kinetic clusters revealed that only the outer part of liver tissue showed an increase of ICG signal. In contrast, deeper parts were characterised by a signal decrease or stable ICG signal over time (Fig. 4d). The signal increase was not only present in liver tissue but also in other regions, which are consistent with blood vessels and regions of increased perfusion (Fig. 4e). The overlaid outline of a manual annotation of the liver in Fig. 4e clearly highlights the heterogeneity of the MSOT signal of ICG within the liver. Sham animals were visually not different from PCI animals with regard to the spatial distribution of kinetic clusters (Supplementary Fig. A5).

Liver-ROIs containing the entire liver tissue were drawn by an expert (Fig. 5a) to compare the results of our new approach with the tissue-oriented analysis. Time-resolved mean, maximum and 95th-percentile signal intensity values within the liver-ROIs were extracted from the pre-processed MSOT images (Fig. 5b). For each of the three features, the time curves from Sham and PCI animals were strongly overlapping, making visual separation of the two treatment groups impossible. After integrating the time derivative values, which did amplify group differences for the signal-oriented analysis, the time curves of Sham and PCI animals were still not visually distinguishable (Supplementary Fig. A6). The results from individual animals of the same treatment group showed a substantial variation, especially for maximum and 95th-percentile values, which was also reflected by the corresponding AUC values (Fig. 5c). No significant differences were found between treatments for the mean and maximum signal intensity values, whereas the effect sizes ranged from low to medium (mean: $p = 0.09$, $g = 0.71$; maximum: $p = 0.22$, $g = 0.42$). Only for the 95th-percentile signal intensity values, the difference between Sham and PCI was just below the level of significance ($p = 0.049$, $g = 0.75$).

To evaluate the impact of tissue heterogeneity on the tissue-oriented analysis, we placed a small circular ROI within the liver tissue. We shifted this ROI multiple times in 5-pixel steps ($\approx$ 0.38 mm) in x- and y-direction (Fig. 5d). The time curves of these ROIs' mean, maximum and 95th-percentile signal intensity values are shown in Fig. 5e. Although all

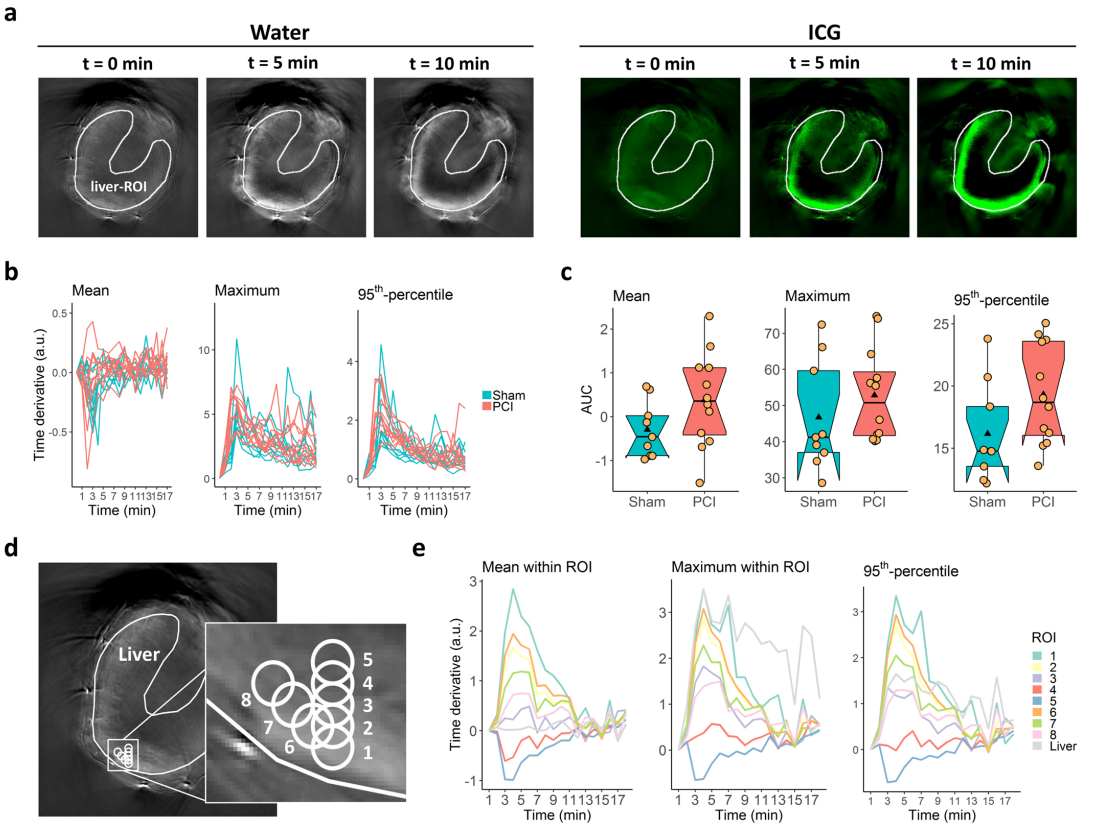


Fig. 5. Tissue-oriented image analysis. (a) Liver-ROIs were drawn manually by an expert in the first time frame of the water channel of the MSOT image stacks (grey coloured images) and used to analyse signal intensity in the ICG channel (green coloured images). (b) The mean, maximum and 95th-percentile time derivative values within the liver-ROIs were extracted from the ICG channel. (c) AUC values of the time curves were calculated for each animal in the two treatment groups and are shown as boxplots for mean, maximum and 95th-percentile signal intensity values, with the black triangle denoting the average value and the yellow dots representing individual data points (mean: $p = 0.09$, $g = 0.71$; maximum: $p = 0.22$, $g = 0.42$; 95th-percentile: $p = 0.049$, $g = 0.75$; p-values were derived by two-tailed Wilcoxon rank sum test; g-values denote Hedges' g). (d) A ROI with a diameter of 10 pixels was drawn in the liver tissue region (ROI 1) and then shifted by 5-pixel steps in the y-direction (ROIs 2–5) or in the x-y-direction (ROIs 6–8) to obtain a set of ROIs within close proximity. (e) The mean, maximum and 95th-percentile signal intensity values from ROIs 1–8 and the liver-ROI were extracted from the pre-processed MSOT image and plotted for the whole time range.

ROIs were in very close vicinity and within the liver, the time curves of the three features followed a gradient from the outer to the inner parts of liver tissue. Furthermore, they turned from a signal increase to a signal decrease at the innermost ROI, demonstrating that signal statistics of specific tissue regions may lead to contradictory results in the tissue-oriented approach and obscure group differences due to spatial inhomogeneities of the signal.

3.4. *Signal-oriented analysis is robust across parameter combinations*

The proposed signal-oriented analysis approach depends on two main parameters: the smoothing factor s and the expected number of clusters k in the k-means clustering algorithm. We screened a range of values for s (from $s = 1$ to $s = 8$) and k (from $k = 2$ to $k = 20$). The impact of s on the resulting main kinetic curves is shown in Supplementary Fig. A7. With $s = 1$, i.e. no smoothing, the resulting kinetic curves were dominated by breathing artefacts. The two lowest levels of smoothing ($s = 2$ and $s = 3$) could not fully compensate for the breathing artefact, especially for $k > 7$. Starting from $s = 4$, breathing artefacts were sufficiently reduced, while higher s resulted in a disproportionate loss of time resolution. The screened value range of k spread from $k = 2$ to $k = 20$, taking into account that large values of k will generally promote overfitting, i.e. the curves of the kinetic clusters become fluctuating also for relatively large values of s, as is apparent from Supplementary Fig. A7.

To evaluate the robustness of our signal-oriented analysis, we compared all parameter combinations with regard to two main properties: (i) high effect size between treatments and (ii) high tolerance towards minor changes of s and k (see Methods Section 2.5 for details on calculations). To take into account the randomised initialisation step of k-means clustering, the analysis was repeated five times for each parameter combination. Regarding property (i), we calculated the mean effect size ($\bar{g}$) and the corresponding standard deviation between AUC values from Sham and PCI treatment groups across the five separate runs for each parameter combination (Fig. 6a and b). The difference between treatment groups in terms of effect size was comparable across a large range of combinations for s and k. Generally, combinations with $s > 1$ and $k > 3$ yielded a mean effect size larger than 0.8, which is considered

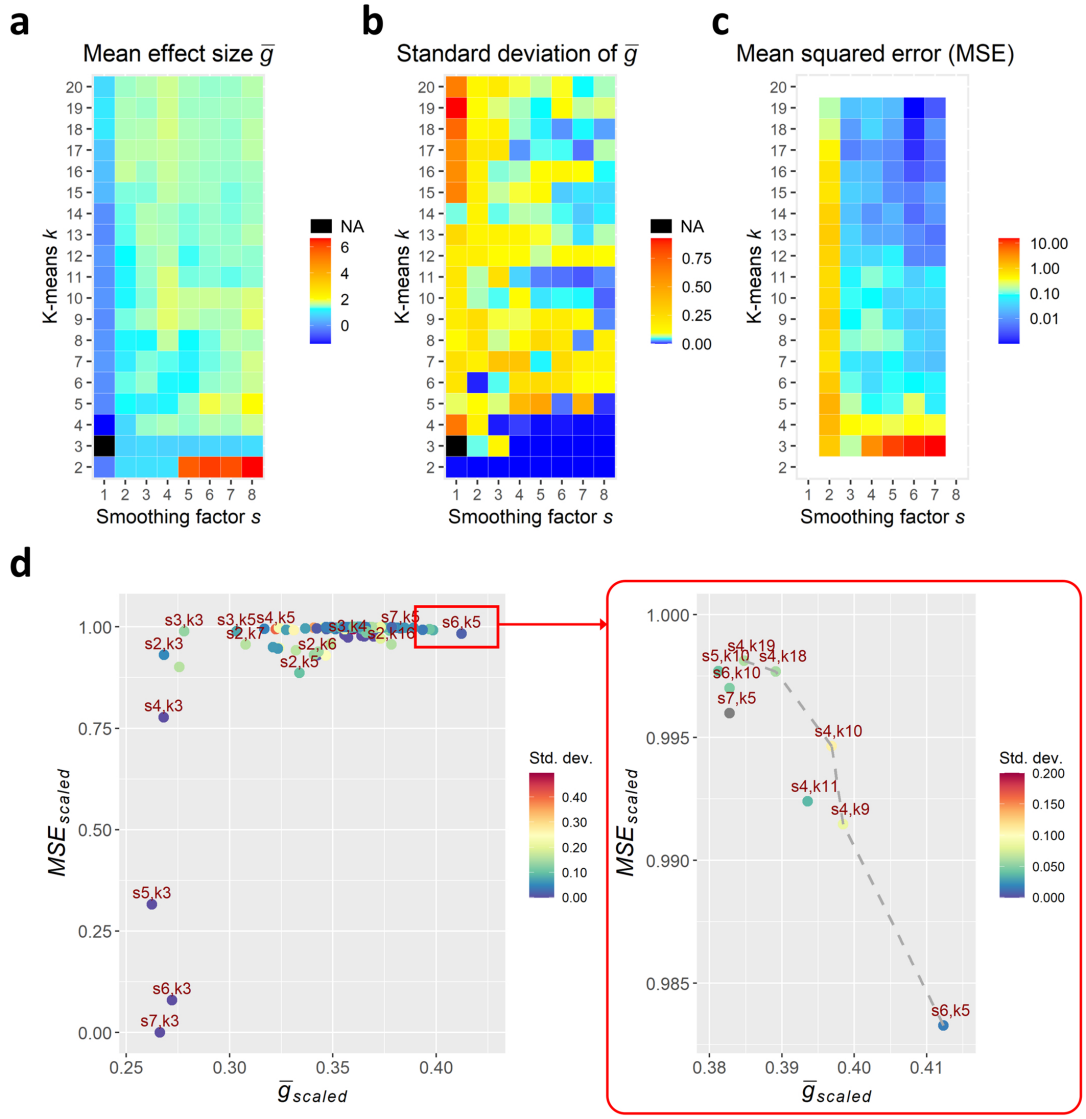


Fig. 6. Parameter estimation for smoothing factor s and k-means k. (a) The mean effect size $\bar{g}$ between Sham and PCI AUC values of five independent runs is highest for $k = 2$ and shows, in addition, several local maxima. (b) Similarly, the corresponding standard deviation of $\bar{g}$ is lowest for small values of k and shows several local minima. (c) The mean squared error (based on the weighted 8-connected neighbourhood of the matrix of $\bar{g}$ values) denotes the robustness towards changes of s and k in the ranges $\pm$ 1 (not calculated for border values because of the uneven number of neighbouring cells). (d) The best parameter combinations of s and k are found at the upper right part in a scatterplot of $\bar{g}_{scaled}$ and MSE_{scaled}. Pareto-optimal parameter combinations are highlighted by the dashed line in the zoomed-in panel on the right.

to represent a large effect, highlighting the robustness of our new approach.

Concerning property (ii), we calculated the MSE of $\bar{g}$ for each parameter combination with regard to $\bar{g}$ of neighbouring parameter combinations (i.e. all possible changes of s and k in the ranges $\pm$ 1; Fig. 6c).

We scaled both $\bar{g}$ and MSE values to the same value range [0,1], and the scaled MSE values were subtracted from 1. The resulting measures are denoted by $\bar{g}_{scaled}$ and MSE_{scaled}. The best parameter combinations are located in the upper right part of a scatterplot of $\bar{g}_{scaled}$ versus MSE_{scaled} (Fig. 6d). We identified multiple Pareto-optimal solutions, i.e. solutions where one feature cannot improve without worsening another, as indicated by the dashed line in the zoomed-in panel of Fig. 6d. These combinations are characterised by comparably high effect size and a relatively low MSE. We decided to use the parameter combination ($s = 4$, $k = 10$) from all Pareto-optimal solutions, representing

intermediate values for s and k. Thus, the loss of details in the kinetic curves with $s = 4$ is less compared to other candidates with $s = 6$ or $s = 7$, whereas using $k = 10$ at the same time offers a suitably high number of kinetic clusters to identify the different tissue types of the cross-section without promoting overfitting.

3.5. Signal-oriented analysis is robust towards slight changes of ROIs

The DL approach for image segmentation was trained on a comparably low number of annotated images. Therefore, we assessed the robustness of our new signal-oriented analysis towards slight changes of the underlying animal-ROIs by performing the analysis for the animal-ROIs derived from the DL-based segmentation as well as for the annotations by the three experimentalists. All animal-ROIs from the three experimenters and the automated segmentation were in visual agreement, as shown in Results Section 3.1. Similarly, the quantitative analysis yielded very similar results for each of the four sets of animal-ROIs (Fig. 7; DL-based segmentation: $p = 8.2 \bullet 10^{-5}$, $g = 1.89$; Experimenter 1: $p = 2.7 \bullet 10^{-5}$, $g = 1.88$; Experimenter 2: $p = 3.8 \bullet 10^{-4}$, $g = 1.86$; Experimenter 3: $p = 8.2 \bullet 10^{-5}$, $g = 1.85$).

3.6. Mcat enables automated and user-friendly application of signal-oriented analysis

To make our new approach easily available and applicable for other researchers, we implemented it in our **M**SOT **c**luster **a**nalysis **t**oolkit *Mcat*, a *Java*-based plugin for *ImageJ* that builds upon *MISA++* [22] and *JGraphT* [23]. It provides an easy-to-use graphical user interface (GUI) for loading data (Supplementary Fig. A8a) and defining parameter sets (Supplementary Fig. A8b).

Data can be imported either manually by specifying input images and, if desired, existing annotations of ROIs, or with the help of a batch importer functionality using a customisable pattern-matcher. If no annotations are provided, the inbuilt DL model will derive the ROIs as described in the Methods Section 2.1. Additionally, custom *Cellpose* models can be imported to adjust the segmentation to other sample types. After the data have been imported, users can adjust the pre-processing, clustering and post-processing parameters. This allows the specification of the channel of interest, the indication of which output data should be generated, the determination of how clustering is applied and which quantitative measurements will be performed. Multiple parameter sets can be defined within one project, which are then automatically executed sequentially.

The software contains components that organise input data and parameters and generates results by applying processing steps. Its two main components are termed *project* and *run* (Fig. 8). A *project* contains all parameters and settings on how to load input data. A *run*, generated from a *project*, includes all necessary functions to perform the analysis with the desired parameter settings. In addition, it contains the data processing steps, which are generated based on the user-defined parameters and the available data. The processing steps are organised in a directed acyclic graph (DAG), where the nodes are the processing steps and edges represent dependencies between these steps. Upon running the analysis, processing steps are executed in topological order. Input and output data are internally organised in a cache structure that is accessed by the processing steps during programme execution. The cache uniquely assigns data, e.g. images, ROIs or results, to the set of parameters and input data. This prevents the repetition of equal workloads and improves the efficiency of the programme. After generating the results, they are automatically saved in the output folder, alongside the parameter file of the *project* that generated the *run*.

The documentation for all parameters used in *Mcat* is provided in Supplementary Information A9. A detailed user manual including installation instructions, the software toolkit and example data is available online at https://github.com/applied-systems-biology/mcat (https://doi.org/10.5281/zenodo.6046031).

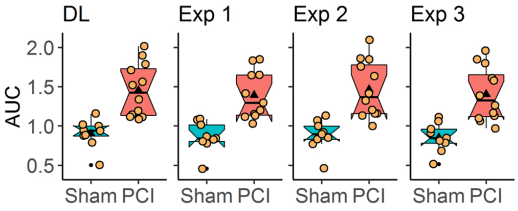


Fig. 7. Robustness of signal-oriented analysis towards slight changes of ROIs. Animal-ROIs were derived by automated segmentation via DL and by manual annotation of three experimenters. The signal-oriented analysis was applied separately for each of the four sets of animal-ROIs. AUC values were calculated for Sham and PCI animals as well as the corresponding p-values and effect sizes, confirming high compliance between the different ROI sets (DL: $p = 8.2 \bullet 10^{-5}$, $g = 1.89$; Experimenter 1: $p = 2.7 \bullet 10^{-5}$, $g = 1.88$; Experimenter 2: $p = 3.8 \bullet 10^{-4}$, $g = 1.86$; Experimenter 3: $p = 8.2 \bullet 10^{-5}$, $g = 1.85$; p-values were derived by two-tailed Wilcoxon rank sum test; g-values denote Hedges' g). The black triangles indicate mean values, and the yellow dots represent individual data points.

4. Discussion and conclusions

This study presented an automated approach to objectively and quantitatively analyse biomarker uptake and distribution from MSOT image data. We utilised and validated our approach by assessing liver function in a well-characterised murine model of sepsis-associated liver failure [16,24], based on the uptake and clearance of the clinically established biomarker ICG [18,25]. We have shown multiple advantages of our signal-oriented approach compared to the widely used tissue-oriented analysis [7–13].

In this study, the tissue-oriented approach as used in its most common form (time-resolved extraction of mean and maximum signal intensities from a ROI) could not detect significant differences between healthy and diseased animals. Only the 95th-percentile signal intensity analysis yielded a group difference that was just below the level of significance ($p = 0.049$). Thus, the results also depended on the analysed intensity feature (Fig. 5c). By placing multiple small ROIs within the region of liver tissue, we could show that confined ROIs are highly sensitive to minor positional changes, and that even those structures that are expected to constitute homogeneous signal kinetics can be characterised by pronounced signal heterogeneity (Fig. 5e). A possible reason for this observation is the limited penetration depth of MSOT. Consequently, quantitative results based on confined ROIs may be misleading, and differences between treatment groups may be overlooked due to tissue heterogeneity, as was the case in the present study for the tissue-oriented analysis. Additionally, the tissue-oriented analysis approach can yield biased results if the locally observed biomarker signal gives rise to the selective placment of ROIs.

In contrast, our signal-oriented approach revealed significant differences in liver function between healthy and diseased animals without estimating the liver areas responsible for drug uptake and removal. This analysis provided a quantitative measure of liver function and enabled the extraction of information about spatial ICG distribution. In contrast to the tissue-oriented approach, our signal-oriented analysis is more objective, since the entire sample is considered, and the regions of photoabsorber uptake are determined by the algorithm itself. DL-based segmentation aids to analyse future data easily without the need for manual annotation, eliminating possible user bias (see Supplementary information A10 for example on application of inbuilt DL-based segmentation model to different MSOT data). However, we want to note that retraining plays an essential role when transferring a neural network to new data and ensures that satisfying results are achieved. Thus, we included both the possibility to use a retrained model for

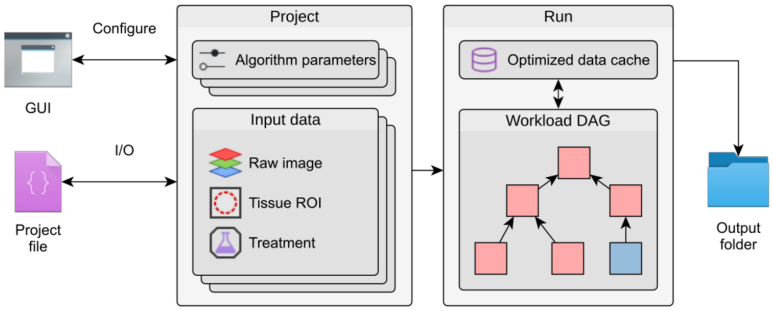


Fig. 8. Internal structure of the *Mcat* software toolkit. *Mcat* organises data in projects that store all parameter sets and groups raw images and ROIs into annotated input data sets. The algorithm parameters can be configured in a graphical user interface (GUI). Upon running a project, a directed acyclic graph (DAG) is generated that models the dependencies between the individual algorithms. In conjunction with the optimised data cache, it prevents duplication of workloads when running multiple parameter sets. The generated result data is written to an output folder together with a project file that includes the parameters and data used in this run.

segmentation, as well as the option to provide manual annotations. Additionally, we could show that our approach is not sensitive to small changes of the ROIs or parameters but provides robust results. At the same time, the complete automation of all processing steps ensures high data throughput. Furthermore, it is not restricted to studying ICG as photoabsorber. It is also possible to analyse other photoabsorbers, as demonstrated for the channel of oxygenated haemoglobin from another MSOT study in Supplementary information A10. For such an analysis, the photoabsorber has to exhibit sufficient resolution in space and time when imaged by MSOT. Our approach is in general limited to the analysis of time kinetics instead of analysing endpoint experiments. The current version of our analysis is also restricted to one anatomical cross-section, with plans to provide multi-slice analysis in future versions of *Mcat*. As the principles of our approach are not specific for MSOT images, we expect it to be transferrable to other types of image data to analyse signal kinetics, e.g. spatio-temporal characterisation of fluorescently labelled molecules from time-resolved microscopy data. However, such an application is beyond the scope of this manuscript and will have to be the subject of future studies.

The fact that we extracted connected areas of comparable signal kinetics shows that the results of our signal-oriented approach are not governed by image noise or signal fluctuations. The analysis of the spatial distribution of these areas was solely based on a visual assessment in this study. However, a quantitative analysis, e.g. by measuring scatteredness, area or homogeneity of specific kinetic clusters, could help to better characterise the liver function of individual animals and to learn the link to particular patterns of the spatial distribution of kinetic clusters. Together with the sample-specific kinetic curves, our approach offers a much more comprehensive analysis of the investigated samples than the tissue-oriented approach and could help to better discriminate severe liver failure from milder impairment of liver function. Such differentiation is a key towards personalised medicine and targeted treatment not only with regard to sepsis [26,27], but also critically ill patients [28], acute liver failure [29] and liver transplantation [30]. Although there already exist methods to assess the ICG plasma disappearance rate as a predictor of organ failure [31], these either involve invasive blood sampling or depend on unimpeded peripheral perfusion, which can be problematic in various cases [26,32]. Non-invasive whole-organ imaging with MSOT, combined with robust and quantitative image analysis, could help identify patients with a critical health status, initiate proper treatment and monitor its effectiveness longitudinally.

Our MSOT cluster analysis toolkit *Mcat* is implemented as an *ImageJ* plugin, which is freely available and easy to use for scientists without programming skills, coming from various research areas (https://github.com/applied-systems-biology/mcat; https://doi.org/10.5281/zenodo.6046031). Furthermore, the software is easy to adapt to various study setups via its user-friendly parameter editor. In combination with the non-invasive imaging modality MSOT, our signal-oriented analysis approach will play an essential role in the objective evaluation of organ function in critically ill patients. It will support the prediction of disease progression and the success of therapy in personalised medicine.

Author Contributions

Z.C., O.S., A.T.P., M.B. and M.T.F. conceptualised the study. W.F., O. S. and A.T.P. designed and performed animal experiments. B.H., R.G., Z. C. and M.T.F. developed image analysis methodology. B.H. and R.G. developed the presented software. B.H. performed the quantitative image analysis. B.H., R.G. and Z.C. validated the image analysis methodology and analysed the data. B.H. and R.G. wrote the original draft and prepared figures. A.T.P., M.B. and M.T.F. supervised the study. M.B. and M.T.F. did project administration and funding acquisition. All authors revised the manuscript and approved it for submission.

Code availability

The presented software toolkit *Mcat*, a user manual and example data are freely available (licensed under BSD 2-clause) at https://github.com/applied-systems-biology/mcat (https://doi.org/10.5281/zenodo.6046031).

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

All MSOT image data, ROI files used in this study as well as code and data used for the DL model training are available at https://doi.org/10.6084/m9.figshare.13567265.v3.

Acknowledgements

This study was funded by the German Research Foundation (DFG), Germany, through the Collaborative Research Centre PolyTarget 1278 “Polymer-based nanoparticle libraries for targeted anti-inflammatory strategies” (Project Z01 to MTF and Project C03 to MB) [grant number 316213987]; the Leibniz ScienceCampus *InfectoOptics* Jena, Germany, which is financed by the funding line Strategic Networking of the Leibniz Association; the Federal Ministry of Education and Research (BMBF), Germany, through the Centre for Sepsis Control and Care (CSCC) [grant number 01EO1502] and the International Leibniz Research School for Microbial and Biomolecular Interactions Jena (ILRS Jena), Germany. ATP acknowledges the Interdisciplinary Centre for Clinical Research, Germany, [grant number AMSP05].

Appendix A. Supporting information

Supplementary data associated with this article can be found in the online version at doi:10.1016/j.pacs.2022.100361.

References

[1] A.B.E. Attia, G. Balasundaram, M. Moothanchery, U.S. Dinish, R. Bi, V. Ntziachristos, M. Olivo, A review of clinical photoacoustic imaging: current and future trends, Photoacoustics 16 (2019), 100144, https://doi.org/10.1016/j.pacs.2019.100144.

[2] G. Diot, S. Metz, A. Noske, E. Liapis, B. Schroeder, S.V. Ovsepian, R. Meier, E. Rummeny, V. Ntziachristos, Multispectral Optoacoustic Tomography (MSOT) of human breast cancer, Clin. Cancer Res. 23 (2017) 6912–6922, https://doi.org/10.1158/1078-0432.CCR-16-3200.

[3] I. Stoffels, S. Morscher, I. Helfrich, U. Hillen, J. Leyh, N.C. Burton, T.C. Sardella, J. Claussen, T.D. Poeppel, H.S. Bachmann, A. Roesch, K. Griewank, D. Schadendorf, M. Gunzer, J. Klode, Metastatic status of sentinel lymph nodes in melanoma determined noninvasively with multispectral optoacoustic imaging, Sci. Transl. Med. 7 (2015), https://doi.org/10.1126/scitranslmed.aad1278.

[4] M. Wildgruber, M. Masthoff, A. Helfen, J. Claussen, A. Karlas, N.A. Markwardt, V. Ntziachristos, M. Eisenblätter, M. Wildgruber, Use of multispectral optoacoustic tomography to diagnose vascular malformations, JAMA Dermatol. 154 (2018) 1457–1462, https://doi.org/10.1001/jamadermatol.2018.3269.

[5] M.J. Waldner, F. Knieling, C. Egger, S. Morscher, J. Claussen, M. Vetter, C. Kielisch, S. Fischer, L. Pfeifer, A. Hagel, R.S. Goertz, D. Wildner, R. Atreya, D. Strobel, M. F. Neurath, Multispectral optoacoustic tomography in Crohn's disease: noninvasive imaging of disease activity, Gastroenterology 151 (2016) 238–240, https://doi.org/10.1053/j.gastro.2016.05.047.

[6] V. Ntziachristos, D. Razansky, Molecular imaging by means of multispectral optoacoustic tomography (MSOT), Chem. Rev. 110 (2010) 2783–2794, https://doi.org/10.1021/cr9002566.

[7] A. Taruttis, S. Morscher, N.C. Burton, D. Razansky, V. Ntziachristos, Fast multispectral optoacoustic tomography (MSOT) for dynamic imaging of pharmacokinetics and biodistribution in multiple organs, PLoS One 7 (2012), https://doi.org/10.1371/journal.pone.0030491.

[8] J. Sharkey, L. Ressel, N. Brillant, L. Scarfe, B. Wilm, B.K. Park, P. Murray, A noninvasive imaging toolbox indicates limited therapeutic potential of conditionally activated macrophages in a mouse model of multiple organ dysfunction, Stem Cells Int. 2019 (2019), https://doi.org/10.1155/2019/7386954.

[9] W. Song, Z. Tang, D. Zhang, N. Burton, W. Driessen, X. Chen, Comprehensive studies of pharmacokinetics and biodistribution of indocyanine green and liposomal indocyanine green by multispectral optoacoustic tomography, RSC Adv. 5 (2015) 3807–3813, https://doi.org/10.1039/c4ra09735a.

[10] S. Morscher, W.H.P. Driessen, J. Claussen, N.C. Burton, Semi-quantitative multispectral optoacoustic tomography (MSOT) for volumetric PK imaging of gastric emptying, Photoacoustics 2 (2014) 103–110, https://doi.org/10.1016/j.pacs.2014.06.001.

[11] Y. Wu, S. Huang, J. Wang, L. Sun, F. Zeng, S. Wu, Activatable probes for diagnosing and positioning liver injury and metastatic tumors by multispectral optoacoustic tomography, Nat. Commun. 9 (2018), https://doi.org/10.1038/s41467-018-06499-1.

[12] Y. Huang, Y. Qi, C. Zhan, F. Zeng, S. Wu, Diagnosing drug-induced liver injury by multispectral optoacoustic tomography and fluorescence imaging using a leucine-aminopeptidase-activated probe, Anal. Chem. 91 (2019) 8085–8092, https://doi.org/10.1021/acs.analchem.9b00107.

[13] L. Scarfe, A. Rak-Raszewska, S. Geraci, D. Darssan, J. Sharkey, J. Huang, N. C. Burton, D. Mason, P. Ranjzad, S. Kenny, N. Gretz, R. Lévy, B.K. Park, M. García-Fiñana, A.S. Woolf, P. Murray, B. Wilm, Measures of kidney function by minimally invasive techniques correlate with histological glomerular damage in SCID mice with adriamycin-induced nephropathy, Sci. Rep. 5 (2015), https://doi.org/10.1038/srep13601.

[14] A. Taruttis, V. Ntziachristos, Advances in real-time multispectral optoacoustic imaging and its applications, Nat. Photonics 9 (2015) 219–227, https://doi.org/10.1038/nphoton.2015.29.

[15] J. Schindelin, I. Arganda-Carreras, E. Frise, V. Kaynig, M. Longair, T. Pietzsch, S. Preibisch, C. Rueden, S. Saalfeld, B. Schmid, J.-Y. Tinevez, D.J. White, V. Hartenstein, K. Eliceiri, P. Tomancak, A. Cardona, Fiji: an open-source platform for biological-image analysis, Nat. Methods 9 (2012) 676–682, https://doi.org/10.1038/nmeth.2019.

[16] F.A. Gonnert, P. Recknagel, M. Seidel, N. Jbeily, K. Dahlke, C.L. Bockmeyer, J. Winning, W. Lösche, R.A. Claus, M. Bauer, Characteristics of clinical sepsis reflected in a reliable and reproducible rodent sepsis model, J. Surg. Res. 170 (2011) 123–134, https://doi.org/10.1016/j.jss.2011.05.019.

[17] S. Seemann, F. Zohles, A. Lupp, Comprehensive comparison of three different animal models for systemic inflammation, J. Biomed. Sci. 24 (2017) 1–17, https://doi.org/10.1186/s12929-017-0370-8.

[18] E. Levesque, F. Saliba, ICG clearance monitoring in ICU patients, Yearb. Intensive Care Emerg. Med. 2009 (2007) 646–657, https://doi.org/10.1007/978-0-387-92278-2_60.

[19] C. Stringer, T. Wang, M. Michaelos, M. Pachitariu, Cellpose: a generalist algorithm for cellular segmentation, Nat. Methods 18 (2021) 100–106, https://doi.org/10.1038/s41592-020-01018-x.

[20] R. Gerst, Z. Cseresnyés, J.P.M.T. Figge. ⟨www.jipipe.org⟩. (Accessed 18 January 2022).

[21] D. Legland, I. Arganda-Carreras, P. Andrey, MorphoLibJ: integrated library and plugins for mathematical morphology with ImageJ, Bioinformatics 32 (2016) 3532–3534, https://doi.org/10.1093/bioinformatics/btw413.

[22] R. Gerst, A. Medyukhina, M.T. Figge, MISA++: a standardized interface for automated bioimage analysis, SoftwareX 11 (2020), 100405, https://doi.org/10.1016/j.softx.2020.100405.

[23] D. Michail, J. Kinable, B. Naveh, J.V. Sichi, JGraphT - a java library for graph data structures and algorithms, ACM Trans. Math. Softw. 46 (2020) 1–33, https://doi.org/10.1145/3381449.

[24] O. Sommerfeld, A. Medyukhina, S. Neugebauer, M. Ghait, S. Ulferts, A. Lupp, R. König, R. Wetzker, S. Schulz, M.T. Figge, M. Bauer, A.T. Press, Targeting complement C5a receptor 1 for the treatment of immunosuppression in sepsis, Mol. Ther. 29 (2021) 338–346, https://doi.org/10.1016/j.ymthe.2020.09.008.

[25] A. Kortgen, M. Paxian, M. Werth, P. Recknagel, F. Rauchfu, A. Lupp, C.G. Krenn, D. Müller, R.A. Claus, K. Reinhart, U. Settmacher, M. Bauer, Prospective assessment of hepatic function and mechanisms of dysfunction in the critically ill, Shock 32 (2009) 358–365, https://doi.org/10.1097/SHK.0b013e31819d8204.

[26] M.F. Kaffarnik, J.F. Lock, H. Vetter, N. Ahmadi, C. Lojewski, M. Malinowski, P. Neuhaus, M. Stockmann, Early diagnosis of sepsis-related hepatic dysfunction and its prognostic impact on survival: a prospective study with the LiMAx test, Crit. Care 17 (2013) R259, https://doi.org/10.1186/cc13089.

[27] M.T. Inal, D. Memiş, M. Kargi, N. Sut, Prognostic value of indocyanine green elimination assessed with LiMON in septic patients, J. Crit. Care 24 (2009) 329–334, https://doi.org/10.1016/j.jcrc.2008.11.012.

[28] S.G. Sakka, Assessment of liver perfusion and function by indocyanine green in the perioperative setting and in critically ill patients, J. Clin. Monit. Comput. 32 (2018) 787–796, https://doi.org/10.1007/s10877-017-0073-4.

[29] U. Merle, O. Sieg, W. Stremmel, J. Encke, C. Eisenbach, Sensitivity and specificity of plasma disappearance rate of indocyanine green as a prognostic indicator in acute liver failure, BMC Gastroenterol. 9 (2009) 91, https://doi.org/10.1186/1471-230X-9-91.

[30] Y. Sun, L. Yu, Y. Liu, Predictive value of indocyanine green plasma disappearance rate on liver function and complications after liver transplantation, Med. Sci. Monit. 24 (2018) 3661–3669, https://doi.org/10.12659/MSM.907783.

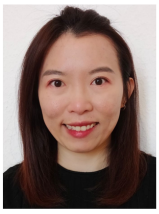
[31] J.J. Vos, J.K.G. Wietasch, A.R. Absalom, H.G.D. Hendriks, T.W.L. Scheeren, Green light for liver function monitoring using indocyanine green? An overview of current clinical applications, Anaesthesia 69 (2014) 1364–1376, https://doi.org/10.1111/anae.12755.

[32] C.A. Pantanali, G.D. Esteban, D. LA, A. W, B. P, A. AM, Lessons learned with the LiMON method of indocyanine green elimination, EC Gastroenterol. Dig. Syst. 4 (2018) 297–304, https://doi.org/10.31080/ecgds.2018.05.00182.

Bianca Hoffmann is working in the field of image-based systems biology. She studied Bioinformatics at the Friedrich-Schiller-University Jena, Germany, and completed her Ph.D. in natural sciences in the research group Applied Systems Biology at the Leibniz Institute of Natural Product Research and Infection Biology e.V. Hans Knöll Institute Jena in 2018. She is now working as a postdoctoral researcher in the group Applied Systems Biology with the research focus being on automated analysis and processing of image data. Her special interest is the extraction of quantitative features from microscopy and tomography image data.

Ruman Gerst is a doctoral student in the group Applied Systems Biology at the Leibniz Institute of Natural Product Research and Infection Biology e.V. – Hans Knöll Institute, Jena, Germany. From 2012–2018 he studied Bioinformatics (B. Sc. and M.Sc.) at the Friedrich-Schiller University Jena, Germany. Since 2018 he is pursuing his PhD with the research focus on quantitative image analysis of infections.

Zoltán Cseresnyés received his university diploma in physics in 1985 in Hungary, at the University of Debrecen. His Ph.D. work in cell biology focused on the confocal microscopy studies of calcium signalling in peripheral neurons. He worked for many years in the USA, Great Britain and Germany. His work interest has been in quantitative microscopy, both from the experimental and analytical point of view. He is currently a postdoctoral fellow at the Hans Knöll Institute in Jena, where he aids the collaborative efforts between microscopists and image analysts. His own research projects focus on label-free image acquisition and analysis methods.



WanLing Foo is currently pursuing her doctoral grade degree in University Hospital Jena, Germany. She is working on targeted drug and gene delivery systems and translational medicine for anti-inflammation. She received her M.Eng. degree in Bioengineering from The University of Tokyo, Japan and B. Eng. degree in Materials Science from Nanyang Technological University, Singapore. Her master focused on targeted gene delivery and systemic gene therapy for anticancer treatments. In between the studies, she has also involved in the research of intrauterine stem cell transplantation for Thalassaemia therapy.

Oliver Sommerfeld is Consultant for Anaesthesiology and Intensive Care. He studied medicine at Friedrich-Schiller University Jena (2004–2011). He completed his specialist training at the Clinic for Anaesthesiology and Intensive Care at the Jena University Hospital (2011–2018). After he received his M.D. in 2013, he completed a research rotation at the Center for Sepsis Control Care (2014–2015). His research is in the field of innate immune response and complement activation in sepsis, transnational medicine and animal models.

Adrian T. Press, studied molecular medicine for his bachelor's and master's degrees at Furtwangen and the Friedrich Schiller University in Jena, Germany. He then obtained his doctoral degree at the medical faculty in Jena researching novel theranostic nanoparticles. During his postdoc, he specialised in researching biophotonic technologies to characterise and model septic organ failure and translate this knowledge to generate novel personalised therapeutics and diagnostics. Since 2021 he is intensifying his research at Friedrich Schiller University as Junior professor for Molecular Medicine of Life-Threatening Infection.

Michael Bauer is Professor and Chair of Anesthesiology and Intensive Care and Chief Executive Director of the Center for Sepsis Control Care (CSCC) at University Hospital of Jena, Germany. He received his M.D. from Saarland University and postgraduate training in molecular biology at Johns Hopkins (1993–1995). After Board Certification in Anesthesiology and Intensive Care, he was visiting professor at the University of North Carolina in Charlotte (1998–1999). Dr. Bauer has published extensively in the fields of molecular mechanisms of sepsis and shock-related organ dysfunction with a focus on biophotonic strategies to visualise liver perfusion and function.

Marc Thilo Figge is Professor of Applied Systems Biology at the Friedrich Schiller University Jena and in the Center for Systems Biology of Infection at the Leibniz Institute for Natural Product Research and Infection Biology in Jena. He received his Ph.D. in Theoretical Physics at the University of Groningen and became Junior Fellow in Systems Immunology at the Frankfurt Institute for Advanced Studies (FIAS) before setting up his own research group in Jena in 2011. His research focuses on image-based systems biology, combining automated quantification of microscopy and spectroscopy data with mathematical modelling and computer simulation.

Supplementary information

Spatial quantification of clinical biomarker pharmacokinetics through deep learning-based segmentation and signal-oriented analysis of MSOT data

Bianca Hoffmann[a, °], Ruman Gerst[a,b, °], Zoltán Cseresnyés[a], WanLing Foo[c,d], Oliver Sommerfeld[c,d], Adrian T. Press[c,d,e], Michael Bauer[c,d], Marc Thilo Figge[a,d,f, *]

[a] Research Group Applied Systems Biology, Leibniz Institute for Natural Product Research and Infection Biology - Hans Knöll Institute (HKI), Beutenbergstr. 11a, 07745 Jena, Germany

[b] Faculty of Biological Sciences, Friedrich Schiller University Jena, Bachstr. 18k, 07743 Jena, Germany

[c] Department of Anesthesiology and Intensive Care Medicine, Jena University Hospital, Am Klinikum 1, 07747 Jena, Germany

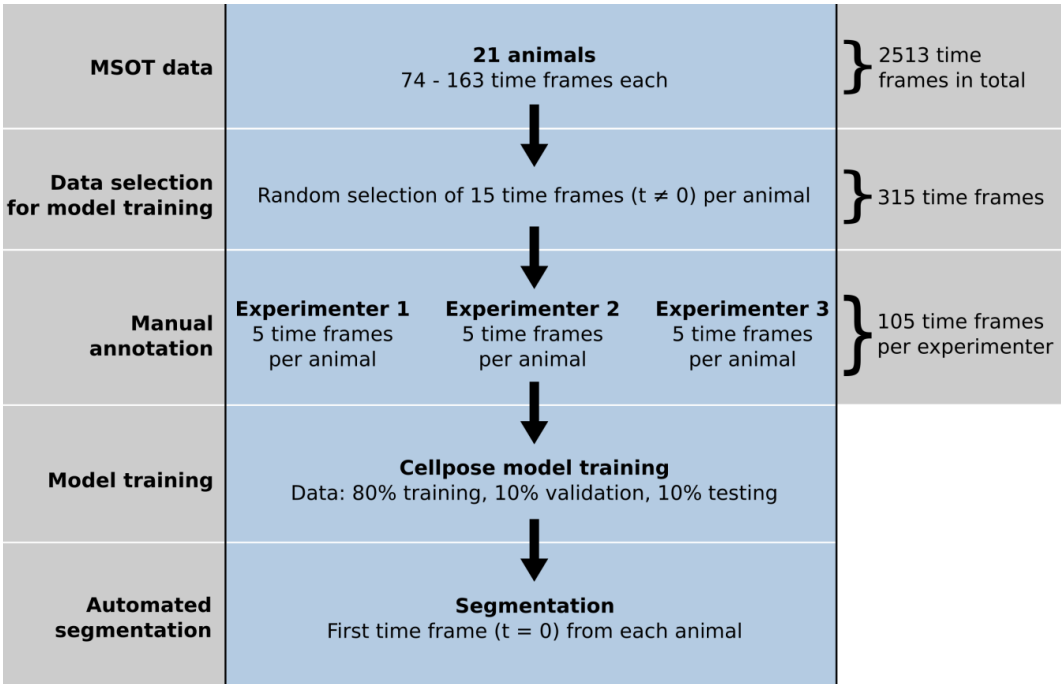
[d] Center for Sepsis Control and Care, Jena University Hospital, Am Klinikum 1, 07747 Jena, Germany

[e] Faculty of Medicine, Friedrich Schiller University Jena, Am Klinikum 1, 07747 Jena, Germany

[f] Institute of Microbiology, Faculty of Biological Sciences, Friedrich Schiller University Jena, Neugasse 25, 07743 Jena, Germany

° Authors contributed equally

* Correspondence: thilo.figge@leibniz-hki.de



Supplementary Figure A1: Schematic of training of deep learning-based segmentation.
Each of the 21 MSOT image stacks consisted of up to 163 time frames with a total of 2513 time frames. For each animal, 15 time frames were selected randomly, excluding the first time frame (t = 0), and distributed among the three experimenters. These 315 time frames were used as ground truth for the training of the Cellpose neural network and were divided into 80% for training, 10% for testing and 10% for validation. The derived model was then used for the final segmentation of the first time frame (t = 0) of each MSOT image stack.

Supplementary Table A2: Software and libraries used in *Mcat.*

Software	Version	URL
Apache Commons	2.6.0	https://commons.apache.org/
Apache Maven	3.6.3	http://maven.apache.org/
ImageJ	1.52r	https://imagej.net/
ImageScience	3.0.0	https://imagescience.org/
Imglib2	5.1.0	https://imagej.net/
FeatureJ	2.0.0	https://imagescience.org/meijering/software/featurej/
Flexmark	0.18.5	https://github.com/vsch/flexmark-java
Jackson (JSON)	2.11.2	https://github.com/FasterXML/jackson
Java	1.8	https://java.com/
JFreeChart	1.5.0	http://jfree.org/
JFreeSVG	3.4	http://jfree.org/
JGraphT	1.3.1	https://jgrapht.org/
IJP-Toolkit	2.1.2	https://github.com/ij-plugins/ijp-toolkit
MorphoLibJ	1.4.1	https://github.com/ijpb/MorphoLibJ/
MTrackJ	1.5.4	https://imagescience.org/meijering/software/mtrackj/
MultiStackRegistration	1.46.2	https://github.com/miura/MultiStackRegistration
RandomJ	2.0.0	https://imagescience.org/meijering/software/randomj/
SciJava	27.0.1	https://scijava.org/
Slf4J	1.7.9	http://www.slf4j.org/
SwingX	1.6.5-1	https://mvnrepository.com/artifact/org.swinglabs/swingx

Supplementary Information A3: Descriptive statistics

Signal-oriented analysis

AUC values

μ_{Sham} = 0.9; σ_{Sham} = 0.19

μ_{PCI} = 1.46; σ_{PCI} = 0.35

Tissue-oriented analysis

AUC of mean signal intensity values

μ_{Sham} = -0.3; σ_{Sham} = 0.64

μ_{PCI} = 0.38; σ_{PCI} = 1.07

AUC of maximum signal intensity values

μ_{Sham} = 46.73; σ_{Sham} = 15.36

μ_{PCI} = 52.83; σ_{PCI} = 12.93

AUC of 95^{th}-percentile signal intensity values

μ_{Sham} = 16.15; σ_{Sham} = 3.96

μ_{PCI} = 19.3; σ_{PCI} = 4.03

Robustness of signal-oriented analysis towards slight changes of ROIs

AUC values deep learning-based segmentation

μ_{Sham} = 0.9; σ_{Sham} = 0.18

μ_{PCI} = 1.45; σ_{PCI} = 0.34

AUC values experimenter 1

μ_{Sham} = 0.86; σ_{Sham} = 0.19

μ_{PCI} = 1.45; σ_{PCI} = 0.36

AUC values experimenter 2

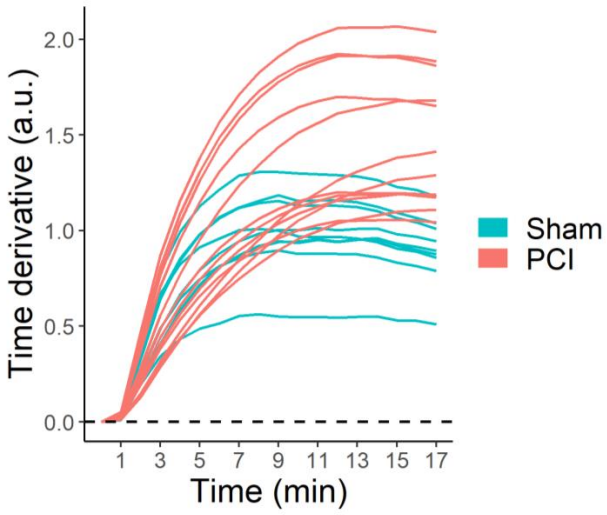
μ_{Sham} = 0.88; σ_{Sham} = 0.2

μ_{PCI} = 1.46; σ_{PCI} = 0.36

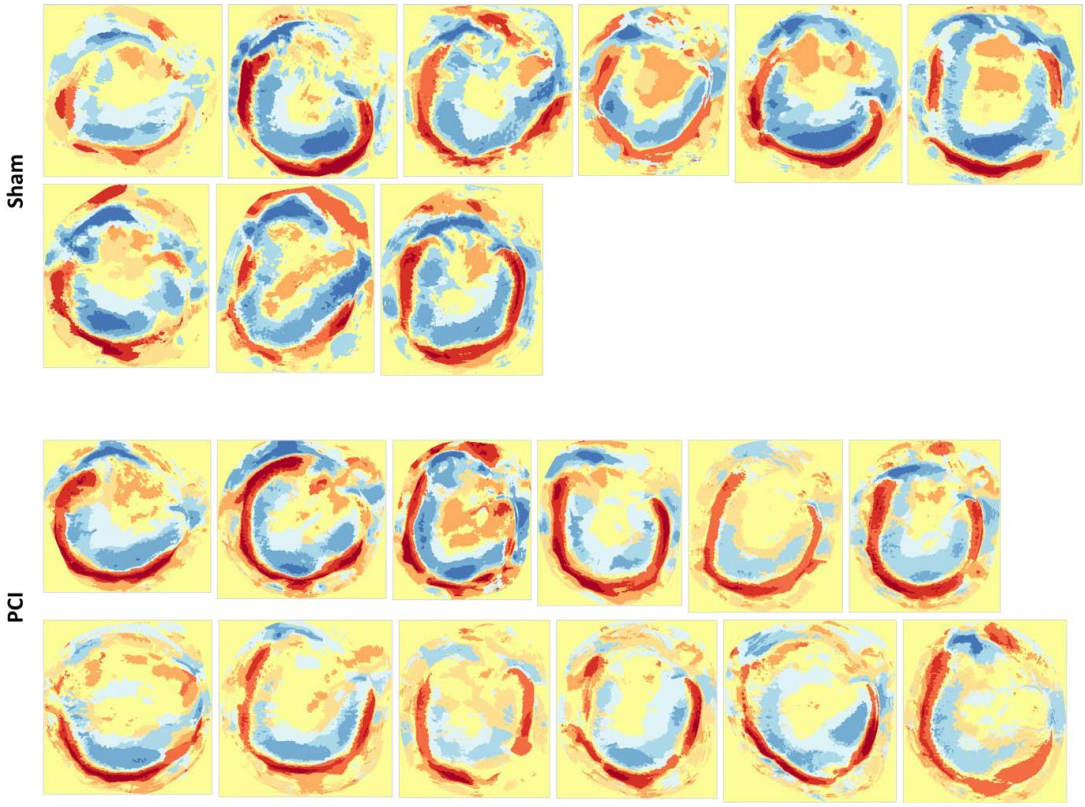
AUC values experimenter 3

μ_{Sham} = 0.85; σ_{Sham} = 0.19

μ_{PCI} = 1.4; σ_{PCI} = 0.34

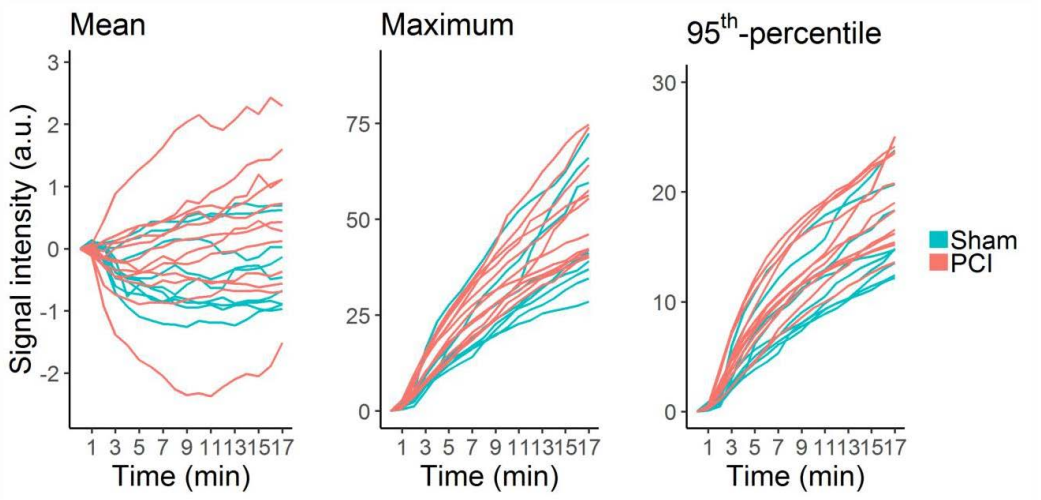


Supplementary Figure A4: Integrated signal of WACs from the signal-oriented analysis.
The time derivative values of the WACs were integrated to obtain the original kinetic shape of the curves for the signal-oriented analysis. The curves of Sham and PCI animals can visually be well distinguished. Seven of the twelve PCI animals show a delayed signal increase and lower plateau values.



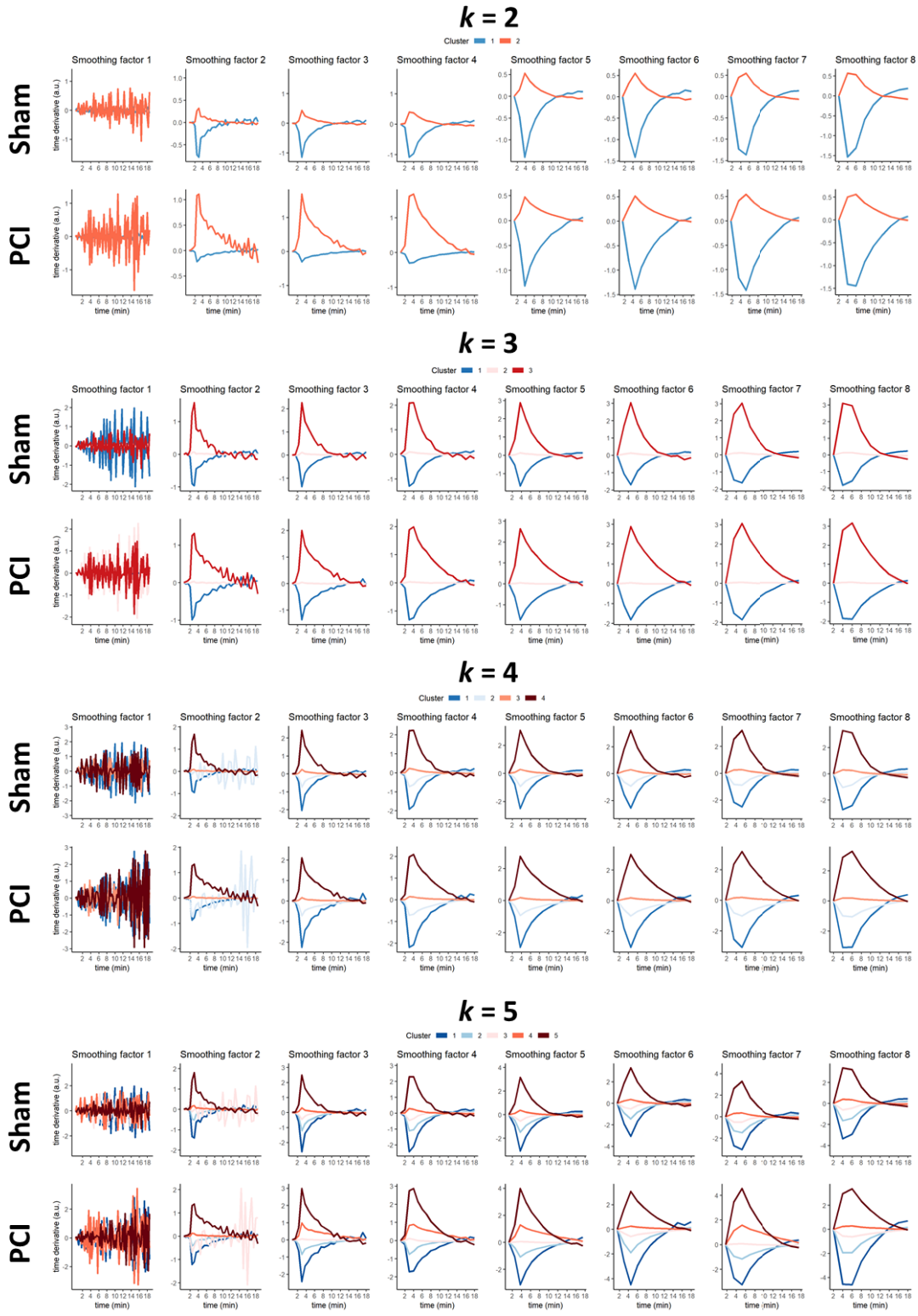
Supplementary Figure A5: Spatial distribution of kinetic clusters.

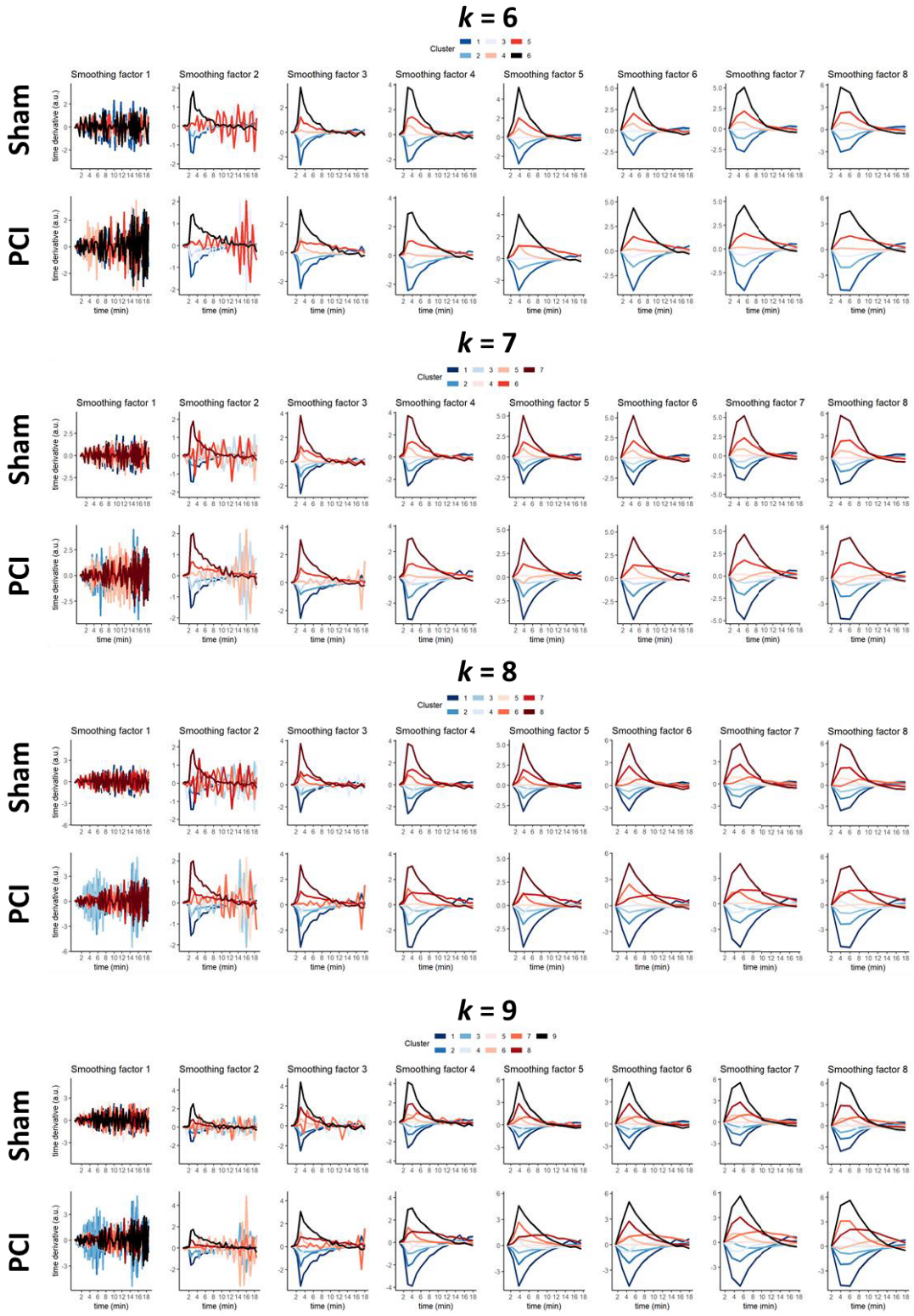
Colour-coded images reveal the biodistribution of ICG. The two top rows show all colour-coded images for Sham animals, the two bottom rows for PCI animals. The biodistribution is visually not different between Sham and PCI animals.

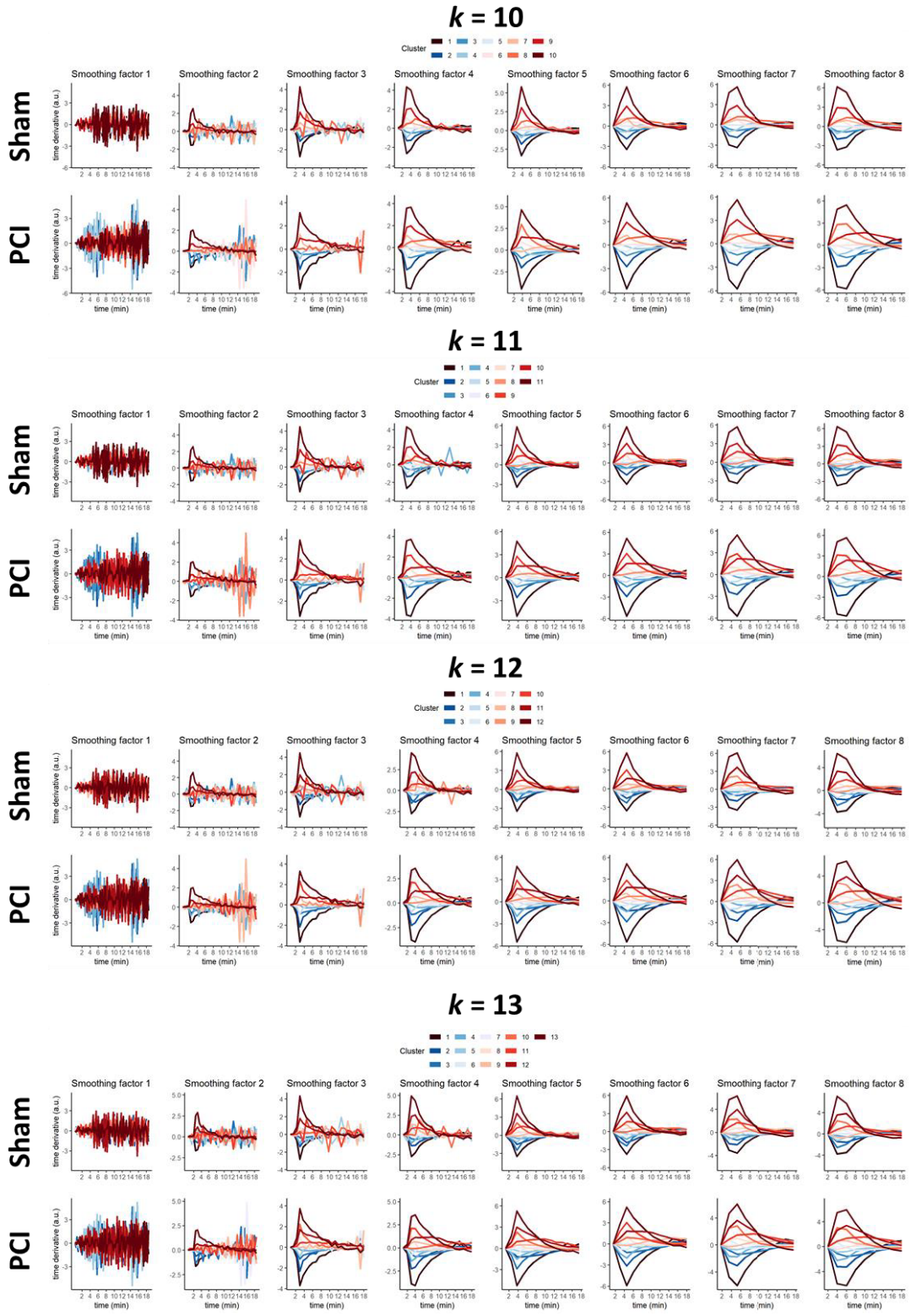


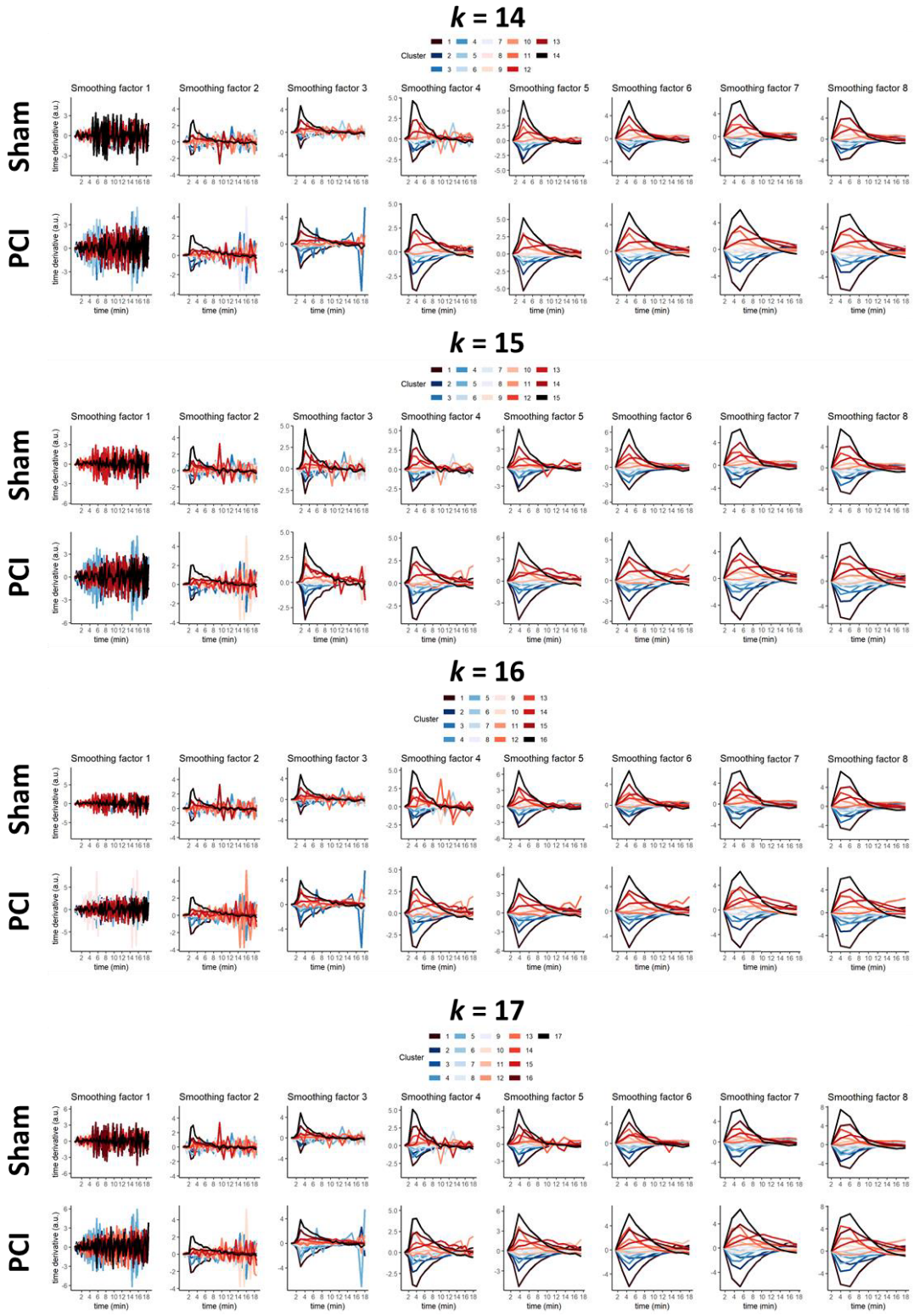
Supplementary Figure A6: Integrated signal of time derivative values from tissue-oriented analysis.

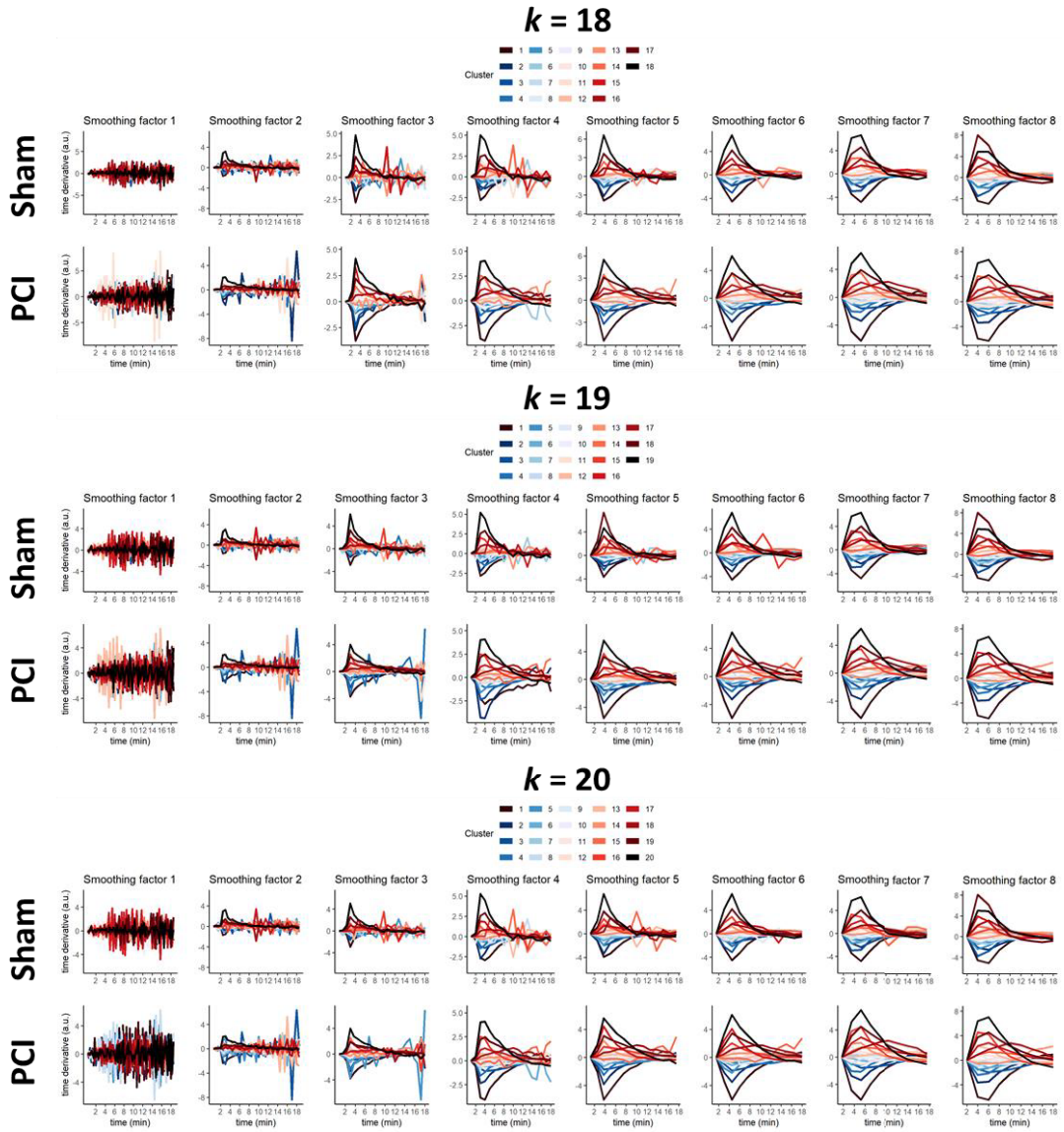
The time derivative values were integrated for the three different intensity features of the tissue-oriented analysis to obtain the original kinetic shape of the ICG signal uptake. The time curves of Sham and PCI animals are strongly overlapping, making visual distinction impossible.


k = 2
Cluster 1 2
k = 3
Cluster 1 2 3
k = 4
Cluster 1 2 3 4
k = 5
Cluster 1 2 3 4 5
Sham
PCI
Smoothing factor 1
Smoothing factor 2
Smoothing factor 3
Smoothing factor 4
Smoothing factor 5
Smoothing factor 6
Smoothing factor 7
Smoothing factor 8
time derivative (a.u.)
time (min)


k = 6
Cluster 1 2 3 4 5 6
k = 7
Cluster 1 2 3 4 5 6 7
k = 8
Cluster 1 2 3 4 5 6 7 8
k = 9
Cluster 1 2 3 4 5 6 7 8 9
Sham
PCI
Smoothing factor 1
Smoothing factor 2
Smoothing factor 3
Smoothing factor 4
Smoothing factor 5
Smoothing factor 6
Smoothing factor 7
Smoothing factor 8
time derivative (a.u.)
time (min)

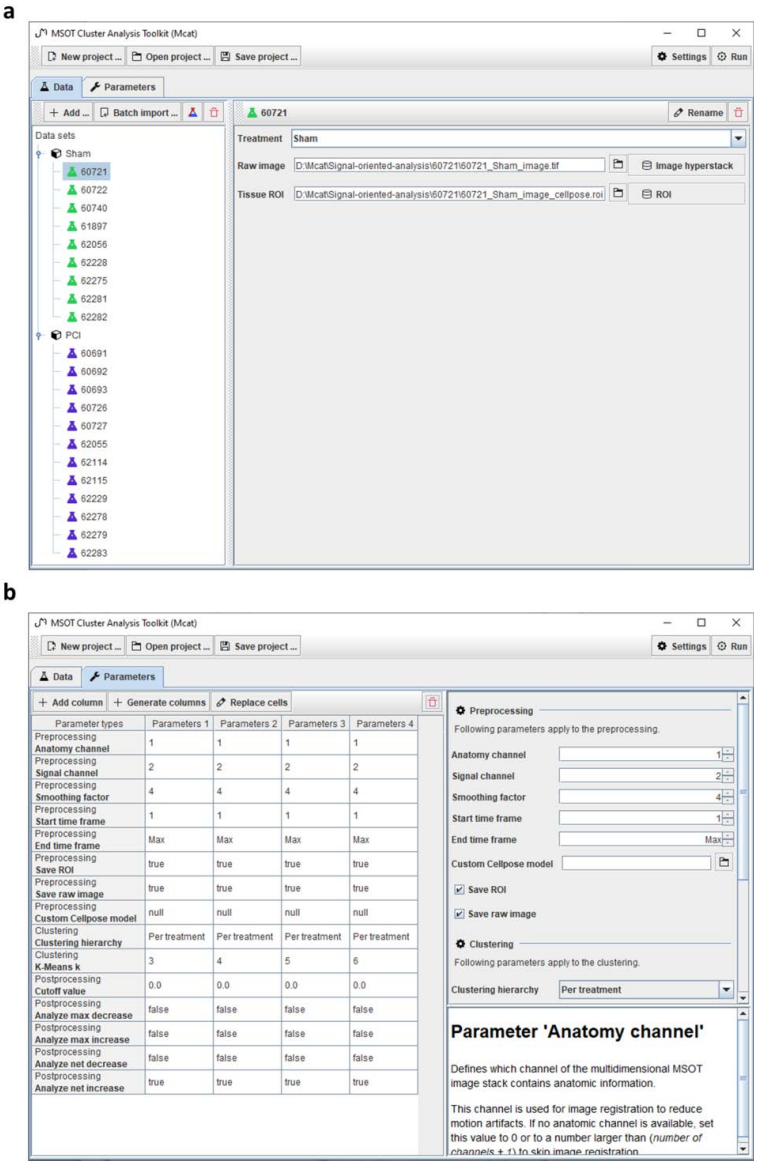

k = 10
Cluster 1 2 3 4 5 6 7 8 9 10
Smoothing factor 1
Smoothing factor 2
Smoothing factor 3
Smoothing factor 4
Smoothing factor 5
Smoothing factor 6
Smoothing factor 7
Smoothing factor 8
Sham
PCI
time derivative (a.u.)
time (min)
k = 11
Cluster 1 2 3 4 5 6 7 8 9 10 11
k = 12
Cluster 1 2 3 4 5 6 7 8 9 10 11 12
k = 13
Cluster 1 2 3 4 5 6 7 8 9 10 11 12 13


k = 14
Cluster
Smoothing factor 1
Smoothing factor 2
Smoothing factor 3
Smoothing factor 4
Smoothing factor 5
Smoothing factor 6
Smoothing factor 7
Smoothing factor 8
Sham
PCI
time derivative (a.u.)
time (min)
k = 15
Cluster
Smoothing factor 1
Smoothing factor 2
Smoothing factor 3
Smoothing factor 4
Smoothing factor 5
Smoothing factor 6
Smoothing factor 7
Smoothing factor 8
Sham
PCI
time derivative (a.u.)
time (min)
k = 16
Cluster
Smoothing factor 1
Smoothing factor 2
Smoothing factor 3
Smoothing factor 4
Smoothing factor 5
Smoothing factor 6
Smoothing factor 7
Smoothing factor 8
Sham
PCI
time derivative (a.u.)
time (min)
k = 17
Cluster
Smoothing factor 1
Smoothing factor 2
Smoothing factor 3
Smoothing factor 4
Smoothing factor 5
Smoothing factor 6
Smoothing factor 7
Smoothing factor 8
Sham
PCI
time derivative (a.u.)
time (min)



Supplementary Figure A7: Curves of kinetic clusters for Sham and PCI treatment with different values of k-means *k*.

The curves of the kinetic clusters as found by pixel-wise clustering with varying values for *k* are shown for the smoothing factor *s* ranging from 1 to 8 and k-means *k* ranging from 2 to 20. Low values of *s* show strong breathing artefacts in the curves, while increasing *s* leads to decreasing temporal resolution. Large values of *k* result in curves with apparent signal fluctuations also for larger values of *s*, which are expected due to overfitting.



Supplementary Figure A8: Graphical user interface of *Mcat* software toolkit.

The main graphical user interface is divided into a *Data* **(a)** and a *Parameters* **(b)** view. In the *Data* view, data can be imported manually or by a batch importer functionality, and imported sample data can be edited. Analysis parameters can be adjusted in the *Parameters* view. A short description of each parameter is shown in the bottom right part of the view when hovering over a parameter name.

Supplementary Information A9: Documentation of *Mcat* parameters

The MSOT clustering analysis toolkit (*Mcat*) comprises three main steps: (i) pre-processing of the MSOT image stacks, (ii) pixel-wise clustering of kinetics and (iii) post-processing of the clustering result. For these steps, the following parameters can be adjusted:

i) Pre-processing

anatomic channel

Defines which channel of the MSOT image stack contains the anatomic information (first channel is indexed as 1). This channel is used for image registration to reduce motion artefacts. If no anatomic channel is available, set this value to 0 to skip image registration.

signal channel

Defines which channel of the MSOT image stack contains the signal that should be analysed. This parameter has to be set to a value between 1 and the number of image channels.

smoothing factor s

Defines how strongly image data is smoothed in the time domain. This parameter is used to reduce breathing artefacts by performing downsampling and averaging over *s* consecutive time frames. Set this value to 1 if the image data shall not be smoothed. Smaller values lead to increased computation time and possibly strong fluctuations in the extracted kinetic curves, while larger values decrease computation time while reducing the time resolution.

start time frame and end time frame

Define which time frames are taken into account. *Start time frame* and *end time frame* have to be in the range of 0 and number of time frames. The stack will be cropped to the time range [*start time frame*, *end time frame*].

save ROI and save raw image

Define if raw data is saved to the output folder. If *save ROI* is enabled, the original or derived ROI file will be stored in the output folder. If *save raw* image is enabled, the original raw image will be stored in the output folder.

custom Cellpose model

If a custom Cellpose model shall be used for the segmentation, specify the path to the model file here. Don't rename Cellpose model files as the name encodes important model information.

ii) Clustering

k-means k

Defines how many kinetic clusters are extracted. This parameter controls how many clusters are used when performing k-means clustering. Smaller values can lead to bad approximation of the

true kinetics present in the image, while larger values can lead to overfitting to confined regions with possibly strongly fluctuating signal intensities.

clustering hierarchy

Defines how samples are grouped when performing k-means clustering. This parameter can be set to one of the three options: *per subject*, *per treatment* or *all in one*.
If *per subject* is used, clustering is performed for each subject individually, resulting in individual kinetic clusters for each subject. This setting can be used to see if subjects show different kinetics. If *per treatment* is used, all subjects from one treatment are grouped together and clustering is performed per treatment. This setting can be used to compare treatments against each other. If *all in one* is used, all subjects are grouped together for clustering. This setting can be used to obtain main kinetic clusters for the whole dataset and examine how the clusters are distributed across samples.

iii) Post-processing

cutoff value

Defines which part of the kinetic curves is used for calculation of area under the curve (AUC) statistics. This parameter can be set to a floating point number between 0 and 1 and controls which portion at the beginning of the curves is excluded from AUC calculation. If this parameter is set to 0, the whole curve is taken into account. If it is set to 0.5, the first half of the curve is omitted. This option can for example be useful to analyse initial dye uptake and dye excretion separately or to limit a study to certain phases of signal development.

analyse max decrease

Defines if the kinetic cluster with the maximum signal decrease should be used for AUC calculation. If it is enabled, a weighted curve is calculated for each subject with the weight being the corresponding pixel abundance value for the kinetic cluster with the maximum signal decrease. These curves are then normalised by the total number of pixels of the respective subjects and the AUC is calculated, taking into account the cutoff value.

analyse net decrease

Defines if all kinetic clusters with a signal net decrease should be used for AUC calculation. If it is enabled, a weighted curve is calculated for each subject with the weights being the corresponding pixel abundance values for all kinetic clusters with a signal net decrease. These curves are then normalised by the total number of pixels of the respective subjects and the AUC is calculated, taking into account the cutoff value.

analyse max increase

Defines if the kinetic cluster with the maximum signal increase should be used for AUC calculation. If it is enabled, a weighted curve is calculated for each subject with the weight being the corresponding pixel abundance value for the kinetic cluster with the maximum signal

increase. These curves are then normalised by the total number of pixels of the respective subjects and the AUC is calculated, taking into account the cutoff value.

analyse net increase

Defines if all kinetic clusters with a signal net increase should be used for AUC calculation. If it is enabled, a weighted curve is calculated for each subject with the weights being the corresponding pixel abundance values for all kinetic clusters with a signal net increase. These curves are then normalised by the total number of pixels of the respective subjects and the AUC is calculated, taking into account the cutoff value.

Supplementary Information A10: Application of DL-based segmentation and signal-oriented analysis to different MSOT data and photoabsorber

To evaluate the applicability of our DL-based segmentation for other MSOT data and our signal-oriented analysis for a photoabsorber other than ICG, we used Mcat to analyse MSOT data that we derived from another study.

In brief, eleven FVB/N mice of mixed gender and aged older than eight weeks were anesthetised and shaved thoroughly for the whole abdomen area using shaver and commercial hair removal cream. After insertion of a tail-vein catheter, mice were placed in a Multispectral Optoacoustic Tomography inVision 256-TF machine (iTheraMedical, Germany) equipped with laser wavelength from 680-980 nm. Mice were anaesthetised with 1.5 to 2 % of isoflurane vaporised in oxygen throughout the preparation and imaging process. After 2 min of baseline image acquisition, 30 µg polyplex micelles loaded with BHQ3 labelled siRNA were injected intravenously through the tail vein catheter. Imaging was continued for 45 min at 6 wavelengths (680 nm, 700 nm, 720 nm, 760 nm, 800 nm, 900 nm) and captured two cross-sectional frames, i.e. liver and kidney. The acquired raw MSOT images were reconstructed by model-based backprojection (filter range: 50kHz to 6.5MHz) using the proprietary software ViewMSOT v3.8.1.04 (iTheraMedical, Munich, Germany), and spectrally unmixed into 4 channels (water, BHQ3, deoxygenated blood, oxygenated blood) by linear regression.

As a proof of principle, we performed the automated segmentation of the liver in the water channel with the inbuilt DL model and compared it to manual segmentation. We then applied the signal-oriented analysis to extract five kinetic clusters for the channel representing oxygenated blood for both DL-based and manual segmentation. Except for the channel of interest, all other settings were kept at their default values.

The comparison of DL-based segmentation and manual segmentation was carried out as described in the main manuscript in section 3.1. The resulting Dice scores for all animals are shown in Fig. A10a. The Dice score for nine out of the eleven animals was comparable to the concordance found between experimenters 2 and 3 in the main manuscript (see Fig. 2 in main manuscript). Two animals reached clearly lower Dice scores. While large parts of the animal outlines were identified correctly in these cases, the upper left region was not segmented properly. This area typically has low resolution of the animal outline because it spatially corresponds to the 90-degree “dead space”, where no ultrasonic detectors are placed due to the need to provide physical access to the sample. Representative examples of manual and DL-based segmentation are shown in Fig. A10b, including one of the cases with a lower Dice score at the very right.

The results of the signal-oriented analysis for DL-based and manual segmentation are shown in Supplementary Figs. A10c and A10d. The extracted kinetic clusters were virtually identical for both DL-based and manual segmentation. Also the AUC values that were calculated for the kinetic clusters reflecting a signal net increase (see section 3.2 of main manuscript for details on calculation) were found to be very similar (Wilcoxon rank sum test $p = 0.89$, Hedges' $g = 0.018$).

These findings demonstrate the general applicability of our inbuilt DL-based segmentation for similar MSOT images and highlight that our signal-oriented analysis does not rely on exact ROI definition but provides results that are robust against the various ways of determining the mouse ROI.

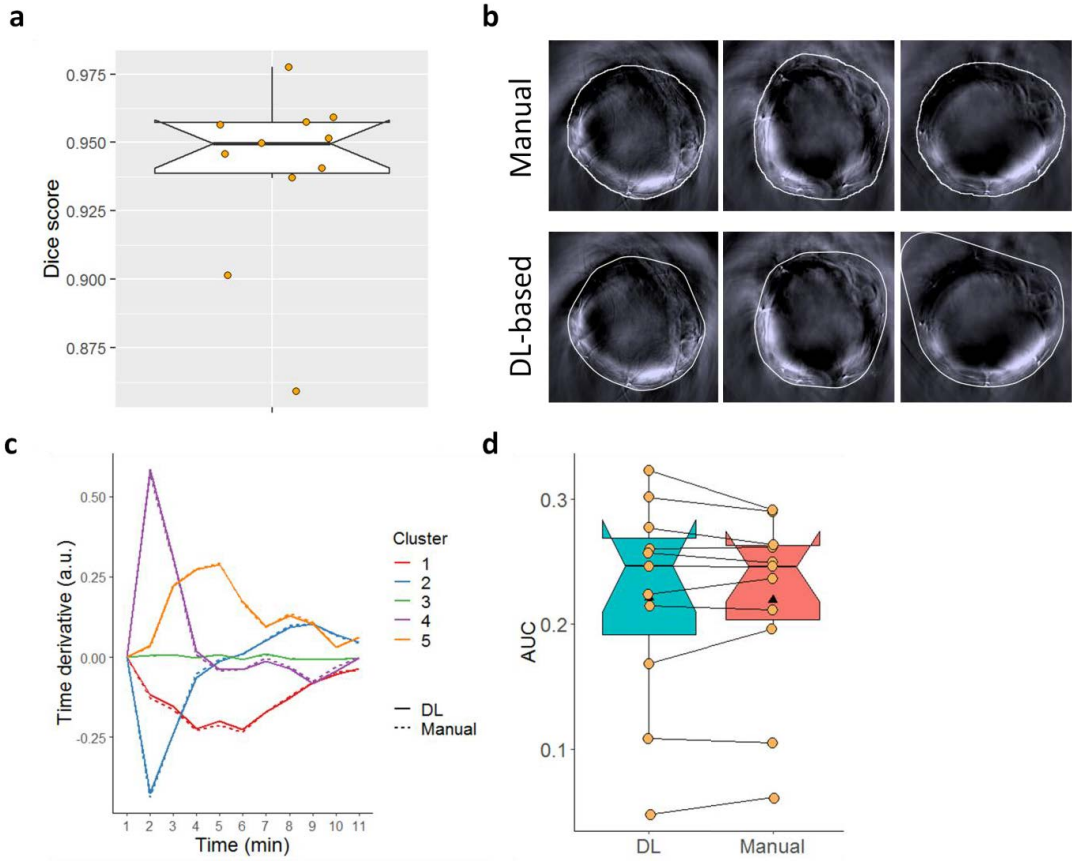


Figure A10: Application of DL-based segmentation and signal-oriented analysis to other MSOT data.

(a) The Dice score between manual and DL-based segmentation shows a high concordance for nine out of the eleven animals. **(b)** This is also supported by the visual comparison of representative example segmentations. In two cases, the upper left animal outline was not segmented properly with the inbuilt DL model (one example shown at the very right). **(c)** The comparison of extracted kinetic clusters and **(d)** corresponding AUC values of curves reflecting a net increase shows similar results for DL-based and manual segmentation (statistical comparison of AUC values: p = 0.89, g = 0.018).

4.2 JIPipe: Visual batch processing for ImageJ

Manuskript Nr. 2

Titel des Manuskriptes: JIPipe: Visual batch processing for ImageJ

Autoren: Ruman Gerst, Zoltán Cseresnyés, Marc Thilo Figge

Bibliographische Informationen: Gerst, R., Cseresnyes, Z. & Figge, M. T. (2022). JIPipe: Visual batch processing for ImageJ. Research Square (*Preprint*)

Der Kandidat / Die Kandidatin ist (bitte ankreuzen)

☐ Erstautor/-in, ☒ Ko-Erstautor/-in, ☐ Korresp. Autor/-in, ☐ Koautor/-in.

Status: zur Publikation in Nature Methods eingereicht

Anteile (in %) der Autoren / der Autorinnen an den vorgegebenen Kategorien der Publikation

Autor/-in	Konzeptionell	Datenanalyse	Experimentell	Verfassen des Manuskriptes	Bereitstellung von Material
Ruman Gerst	70 %	35 %	30 %	40 %	0 %
Zoltán Cseresnyés	20 %	55 %	60 %	40 %	0 %
Marc Thilo Figge	10 %	10 %	10 %	20 %	100 %
Summe:	100 %	100 %	100 %	100 %	100 %

_____ _____

Unterschrift Kandidat/-in Unterschrift Betreuer/-in (Mitglied der Fakultät)

JIPipe: Visual batch processing for ImageJ

Marc Figge (Thilo.Figge@leibniz-hki.de)
Leibniz Institute for Natural Product Research and Infection Biology https://orcid.org/0000-0002-4044-9166
Ruman Gerst
Leibniz Institute for Natural Product Research and Infection Biology https://orcid.org/0000-0002-0723-6038
Zoltan Cseresnyes
Leibniz Institute for Natural Product Research and Infection Biology

Article

Keywords:

Posted Date: May 11th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1641739/v1

JIPipe: Visual batch processing for *ImageJ*

Ruman Gerst[1,2,#], Zoltán Cseresnyés[1,#], Marc Thilo Figge[1,3,*]

[1] Applied Systems Biology, Leibniz Institute for Natural Product Research and Infection Biology – Hans Knöll Institute (HKI)

[2] Faculty of Biological Sciences, Friedrich-Schiller-University Jena, Germany

[3] Institute of Microbiology, Faculty of Biological Sciences, Friedrich-Schiller-University Jena, Germany

[#] These authors contributed equally

[*] Correspondence should be addressed to thilo.figge@leibniz-hki.de

Abstract

The continuous development of new microscopy techniques requires the parallel evolution of image analysis workflows. *ImageJ* provides a high level of accessibility to bioimage processing, which is still impeded by the necessity of developing scripts to achieve reproducibility, and to comply to the FAIR principles. We provide a visual language termed *JIPipe* that allows the construction of an *ImageJ* workflow purely by designing a flowchart. We already included over 1000 functions from *ImageJ* and its plugins. In return, *ImageJ* is extended with custom-designed algorithms, thus forming a symbiotic relationship with *JIPipe*. Our software includes a fully reproduceable and standardized project format, zero-cost scalability of pipelines, as well as automated data saving into an open format. *JIPipe* was already utilized to solve numerous demanding image analysis tasks, showcasing its wide applicability and adaptability. *JIPipe* contributes towards making bioimage analysis more accessible, thereby fostering collaborations between experimentalists and computer scientists.

Introduction

ImageJ[1] is a widely used tool for bioimage analysis[2] due to its interactive analysis workflow and high extensibility via plugins. Its intuitive graphical user interface (GUI) makes functions easily accessible, even to unexperienced users, while still providing more advanced operations for experts. However, the interactive approach of *ImageJ* has currently two major shortcomings: (1) *ImageJ* does not provide means to ensure analysis in accordance with the FAIR (Findability – Accessibility – Interoperability – Reusability) principles[3], where all processing steps and parameters would be tracked automatically rather than relying on researchers' manual notes on all executed steps and their parameters. (2) *ImageJ* does not provide batch processing via the GUI making the analysis of larger projects with hundreds or thousands of datasets time-consuming and error-prone, as every step needs to be executed manually via the GUI. To solve these issues, *ImageJ* includes a macro language that can be used to write fully automated and reproducible pipelines of all functions available inside the GUI. A major disadvantage of such a scripting language is that the development of custom pipelines requires programming experience. Additionally, to comply with the FAIR principles, there is a significant amount of code required to be implemented that reads and manages images and metadata, keeps track of these during the analysis, and exports the results in the desired

format. Especially in comparison with the *ImageJ* GUI, writing macros is excessively complex and non-accessible for researchers without programming experience.

In recent years, visual programming languages have been developed in various scientific fields to conveniently solve the aforementioned shortcomings of scripting languages. A well-known example is *KNIME*[4] that is an open source, visual data analysis environment focusing on general data processing, while community-developed functions handle images. Another example of a visual programming language is available in *Icy*[5], which provides functionality similar to *KNIME*, whilst focusing exclusively on image analysis and providing methods that allow the application of *ImageJ* commands. Furthermore, *CLIJ*[6] is a visual language that enables interactive utilization of GPU-based pipelines. Visual languages are as well implemented in commercial languages, for example *Apeer*[7], applying the concept of customizing machine learning algorithms for image processing. Outside image analysis applications, *Galaxy*[8] provides a cloud-based visual language for genome sequencing analysis, where web-based image analysis functionality is currently being implemented. In the area of education, *Scratch*[9] became popular as a graphical software for children to learn programming in an accessible way.

All visual programming languages have in common that they provide a GUI for creating a pipeline of nodes that have one or multiple inputs and outputs. These allow for the integration of a node into the pipeline that eventually realizes the automated analysis by the execution of functions in the right order. Such graphical platform is not yet available in *ImageJ*. Therefore, we developed the Java Image Processing Pipeline (*JIPipe*) as a visual programming language that is specifically designed for the *ImageJ* software ecosystem. All functionality that is familiar to *ImageJ* users is as well available in *JIPipe* in form of nodes, including operations on images, regions of interest and tables. *JIPipe* also extends to adapted functionalities that would not be easily achievable in native *ImageJ* methods, including batch visualization, advanced plotting, table processing and automated data export. In other words, *JIPipe* integrates visual programming into the existing *ImageJ* software ecosystem providing a user-friendly way to create reproducible and fully automated pipelines based on *ImageJ*. In addition, *ImageJ* can utilize algorithms and pipelines made for *JIPipe* via the GUI or via macros in analogy to any other plugin. The simplicity of scaling, pipeline building, compartmentalization and automated saving features make *JIPipe* a well-suited environment for *ImageJ* users to create reproducible and scalable workflows. *JIPipe* fills a niche by bridging the gap between researchers[10] with limited programming experience and the advancing development of modern methods of automated image processing. In particular, popular *ImageJ* plugins including *MorphoLibJ*[11], *FeatureJ*[12], *CLIJ*[6], *Multi-Template-Matching*[13], *OMERO*[14], and *Bio-Formats*[15] can be accessed via *JIPipe*, as well as *Cellpose*[16] to perform deep learning by visual programming. Furthermore, *JIPipe* comes with nodes for *Python*[17], *Jython*[18], *R*[19], and *ImageJ* macro scripts, thus extending the flexibility of workflow development for experienced users with programming skills. JIPipe has already been successfully applied to solve numerous image analysis tasks, including recently published works on nematode activity analysis[20] and the quantification of multispectral optoacoustic tomography (MSOT) images[21], thus underlining its flexibility and wide applicability.

Results

Symbiosis of *ImageJ* and *JIPipe*. The relationship between *ImageJ* and *JIPipe* is symbiotic in that the two platforms share their functionality (see **Fig. 1**). *JIPipe* incorporates a growing set of already more than 1000 functions from *ImageJ* and encapsulates them into features that are suitable for the visual batch processing environment. GUI elements, including manager windows for images, tables, and regions of interest (ROI), are transformed into data objects to lift limitations on the number of instances and to

improve the processing speed. Operations on these data, available as menu commands in *ImageJ* are encapsulated into nodes with standardized interfaces for data transfer and parameters. This comprises common image processing utilities, including thresholding, extraction of measurements and ROI, as well as functions provided by popular plugins, including *MorphoLibJ*[11], *FeatureJ*[12], *Multi-Template-Matching*[13], *OMERO*[14], and *Bio-Formats*[15]. We also added support for *CLIJ*[6] to allow processing of images on graphics cards with the benefit of significantly increasing the performance. Our visual programming approach comes with an easy-to-learn interface, and automated and standardized (meta-)data management, allowing even non-programmers to connect all these familiar functions in batch processing pipelines. *JIPipe* introduces functionality currently not present in *ImageJ*, including extended visualization methods designed for batch processing, table processing algorithms, nodes for data and metadata management, plotting with the widely used *JFreeChart*[22] library, and support for deep learning via *Cellpose*. To extend the flexibility of workflow development, our software also allows users to embed *Python*, *Jython*, *R*, and *ImageJ* macro scripts. Any functionality designed for *JIPipe* is not exclusive to our software, as any compatible node is automatically made available to be utilized as an *ImageJ* command. Consequently, *ImageJ* benefits from *JIPipe*-exclusive operations that include improved visualization functions, table processing algorithms, plotting, access to *Python* and *R*, as well as Cellpose-based segmentation. This symbiotic relationship additionally simplifies the development of *ImageJ* plugins, as developers can target the *JIPipe* API to develop a library usable in both *ImageJ* and *JIPipe*.

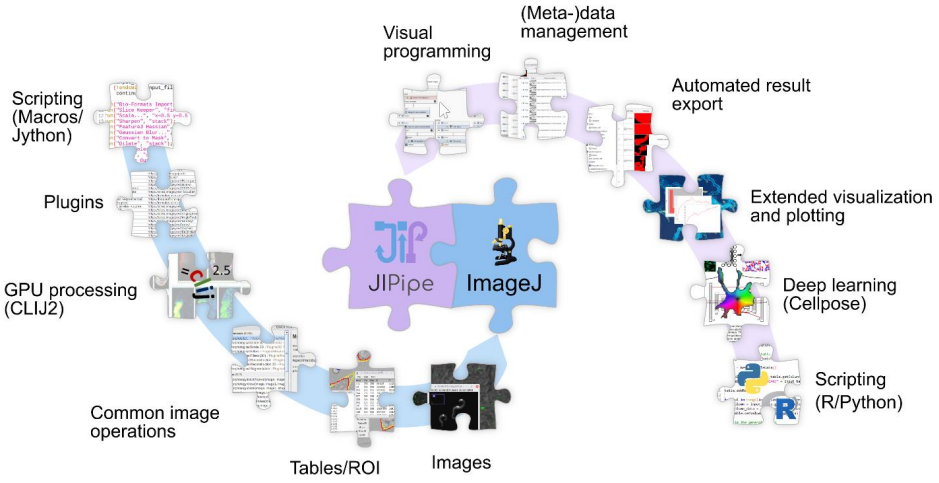


Figure 1 | *JIPipe* and *ImageJ* have a symbiotic relationship where both tools gain access features from each other. *JIPipe* encapsulates *ImageJ* features, including the manual management of images, tables, and regions of interest into a visual programming language with automated data management and export. Operations on these data types are available as nodes and include basic image processing algorithms, and methods from popular plugins, e.g., *CLIJ* for utilizing GPU. *ImageJ* can utilize *JIPipe* functionalities and has access to extended algorithms for visualization and plotting, and integration of *Cellpose*. *JIPipe* can execute *ImageJ* macros and *Jython* scripts, while *JIPipe* provides support for *R* and *Python* scripts.

JIPipe is an easy-to-use alternative to programming *ImageJ* macros, allowing to develop analysis pipelines within a GUI that are the visual counterpart of macros (see **Fig. 2**). *JIPipe* features include automated batch processing, parallelization, and result export. The current set of over 1000 available operations can be extended via plugins that are either provided as Java library or can be created by simply exporting a

pipeline as a new node type (see **Supplementary information 1.1** for an overview). As our software already comes with nodes for scripting languages, a background in Java programming is not needed to integrate plugins and advanced operations not yet encapsulated into nodes. *JIPipe* stores user-created plugins and pipelines in an open and standardized format (see **Supplementary information 1.2** and https://www.JIPipe.org/ for specifications), making it easy to share workflows with the community on a level comparable to *ImageJ* macros.

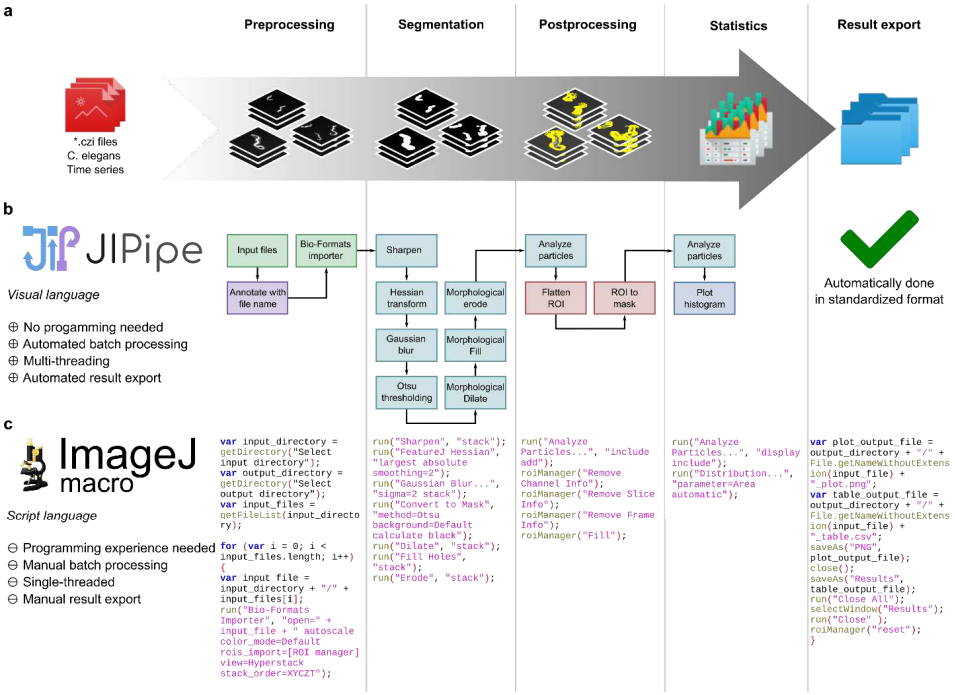


Figure 2 | Comparison of a batch analysis workflow designed with *JIPipe* versus an equivalent *ImageJ* macro. (**a**) The analyzed image data are time series of 2D+time images showing the moving nematode *C. elegans*[20]. This particular workflow extracts the total area covered by *C. elegans* over time from the image files. (**b**) *JIPipe* flowchart that contains the same steps and edges as a *JIPipe* project that applies the analysis (see **Supplementary information 5d**). *JIPipe* automatically saves all results in a standardized format (green check mark). (**c**) *ImageJ* macro code that applies functionality equivalent to the *JIPipe* flowchart.

Hallmarks of *JIPipe* by representative applications. *JIPipe* applications share a set of common features, including file operations, data management and annotations, table processing, visualization, plotting, and saving. The graphical workflow editor allows the arrangement of parameterized nodes into compartmentalized pipelines (**Supplementary Figures 3.1-3.7**) that implement zero-cost scalability due to a table-based "one node -- multiple data" (meta-)data management (**Supplementary Figure 3.7**). Compartments are logical units that can be fully flexibly arranged within a dedicated interface, although they are often arranged according to the preprocessing–analysis–postprocessing–visualization workflow

(**Supplementary Figures 2.1–2.5, Supplementary information 3, Supplementary Figures 3.1-3.6**). Typically, the preprocessing compartment is responsible for file loading and image quality enhancement, as well as channel separation of multichannel images. The analysis compartment carries out the segmentation, masking, and ROI correlation tasks in a channel-specific or conditional way. The outcome is passed on to the postprocessing and visualization compartments (**Supplementary information 2.1-2.5**).

We exemplified the hallmarks of developing image analysis workflows on a set of representative and diverse biological systems that are summarized in **Figure 3**. For instance, the analysis of microfluidic droplets succeeded at characterizing the extent of bacterial growth under various environmental conditions set up inside these picoliter bioreactors (**Fig. 3 row 1**)[23]. Determining the organ-wide distribution of nanocarrier-delivered drugs into the liver of the mouse was the goal in the next example (**Fig. 3 row 2**)[24]. The motility ratio of nematodes characterized the survival of earth worms that consumed soil-beneficial fungi either with or without endosymbiotic bacteria (**Fig. 3 row 3**)[20]. 3D analysis of big volume data was applied to characterize the morphology and spatial distribution of glomeruli in mouse kidneys (see **Fig. 3 row 4**)[25]. Confrontation assays between immune cells and pathogens were quantified by calculating phagocytic measures (**Fig. 3 row 5**)[26]. The various types of image data generated for all these representative biological systems were analyzed in *JIPipe* by exploiting the following features:

Deep learning–based image segmentation. The identification of cell-like objects in both transmitted light (TL) and fluorescence-based images was simplified with the advent of the *Cellpose* framework[16]. *JIPipe* adopted this technique by encapsulating both the training and application of *Cellpose* into nodes, as well as Python environment management tools that allow its fully automated setup. The node extracts the gradient flows (**Fig. 3 A-1 top left, Fig. 3 A5**), the probability map (**Fig. 3 A-1 bottom right**) and the segmented ROIs (**Fig. 3 D-1, D-5**). These outcomes of the analysis indicate that the *JIPipe* integration worked well for TL images of microfluidic droplets (see **Fig. 3 row 1**; **Supplementary information 2.1**) as well as for images of confrontation assays (see **Fig. 3 row 5**; **Supplementary information 2.3**).

Multidimensional image analysis and object tracking. Nodes dedicated to handling multidimensional datasets were utilized when analyzing nanocarrier-based drug delivery kinetics in intravital microscopy of the liver (see **Fig. 3 B-2**; **Supplementary information 2.2**), time series TL data of live nematodes (see **Fig. 3 B-3**; **Supplementary information 2.4**), and the analysis of big volume 3D image stacks as shown in the example of analyzing light-sheet microscopy data of the kidney (**Figure 3 B-4; Supplementary information 2.5**). To this end, these include operations for automated splitting and merging of multi-dimensional data, as well as functionality provided by existing *ImageJ* plugins.

Label-free and fluorescence-based image analysis. JIPipe supplies a complete set of tools to analyze TL and fluorescence-based microscopy data. Hessian filtering was utilized to identify the outline of microfluidic droplets (**Figure 3 C-1, Supplementary information 2.1**), nematodes (**Figure 3 C-3, Supplementary information 2.4**), and unlabeled macrophages in confrontation assays (**Figure 3 C-5, Supplementary information 2.3**). *JIPipe* provides all basic analysis nodes, including blurring, global and local thresholding, morphological operations, watershed algorithms, image normalization and illumination correction, as well as ROI-handling. These tools were applied during the analysis of bacterial growth in microfluidic droplets (**Figure 3 C-1, Supplementary information 2.1**), liver-targeted drug delivery by nanoparticles (**Figure 3 C-2, Supplementary information 2.2**), glomeruli identification and quantification in kidney light-sheet microscopy images (**Figure 3 C-4, Supplementary information 2.5**), as well as labelled fungal spore segmentation (**Figure 3 C-5, Supplementary information 2.3**).

ROI tracking and manipulation. More complex segmentation tasks are also supported by nodes designed for ROI operations including functions provided by *ImageJ* and additional nodes that make use of *JIPipe's* exclusive functionality. These features are utilized in examining i) the extent of bacterial growth inside microfluidic droplets (see **Figs. 3 D-1**, **Supplementary information 2.1**), where the growth area search had to be limited to within the droplet region; ii) the number of dye-loaded liver sinusoidal endothelial cells (LSECs; see **Fig. 3 D-2**, **Supplementary information 2.2**), where masking operations limited the search to be carried out only in the immediate vicinity of the sinusoids; iii) the total area touched by migrating nematodes (see **Fig. 3 D-3**, **Supplementary information 2.4**) where the individual nematode ROIs were merged into the footprint that characterized the motility of the worms; iv) the kidney light-sheet images where ROI-merging via a 3D-connected-components algorithm allowed the volumetric characterization of kidney status under physiological and pathological conditions (see **Fig. 3 D-4**, **Supplementary information 2.5**); v) the distribution of phagocytosed fungal conidia where the ROIs describing the fungal spores were examined for their overlap with those of the macrophages to identify phagocytosed and adherent fungi (see **Fig. 3 D-5**, **Supplementary information 2.3**).

Advanced statistical operations. JIPipe extends the functionality of *ImageJ* with operations for filtering, merging, and manipulating tabular data via user-friendly nodes as well as *Python* and *R* scripts. These were used to calculate percentage growth area of bacteria in microfluidic droplets to identify growth and no-growth classes of the samples (see **Fig. 3 E-1**, **Supplementary information 2.1**). The total footprint covered by a worm divided by the area of the worm at each time point characterized the viability of the nematodes, here showing an example of a low-motility and a high-motility nematode (ratio = 1.3 and 9.0, respectively) (see **Fig. 3 E-3**, **Supplementary information 2.4**). The volumes of individual glomeruli were calculated from the axially merged ROIs in the kidney and plotted in the vicinity of the object (see **Fig. 3 E-4**, **Supplementary information 2.5**). Finally, the calculation of various phagocytic measures from the number of overlapping ROIs was made easy by per-row table manipulations, as demonstrated for the phagocytosis ratio (see **Fig. 3 D-5**, **Supplementary information 2.3**).

Automated plotting. Analysis results of a fully or partially executed workflow can be plotted automatically as supported by a set of dedicated nodes. These provide many graph formats that can be further manipulated in a GUI or via the node parameters. The delivery of nanoparticle-linked drugs to the hepatocytes was described by the kinetic curve of the cargo-linked average fluorescence value (see **Fig. 3 F-2**, **Supplementary information 2.2).** The instantaneous motility of individual worms was calculated and plotted for the duration of the observation (see **Fig. 3 F-3**, **Supplementary information 2.4**), whereas a histogram plot of the glomeruli volume distribution characterized the health status of the kidney sample based on its glomeruli number and size (see **Fig. 3 F-4**, **Supplementary information 2.5**).

These examples demonstrate that *JIPipe* greatly complements *ImageJ* by simplifying the management of a multitude of complex image analysis and management tasks.

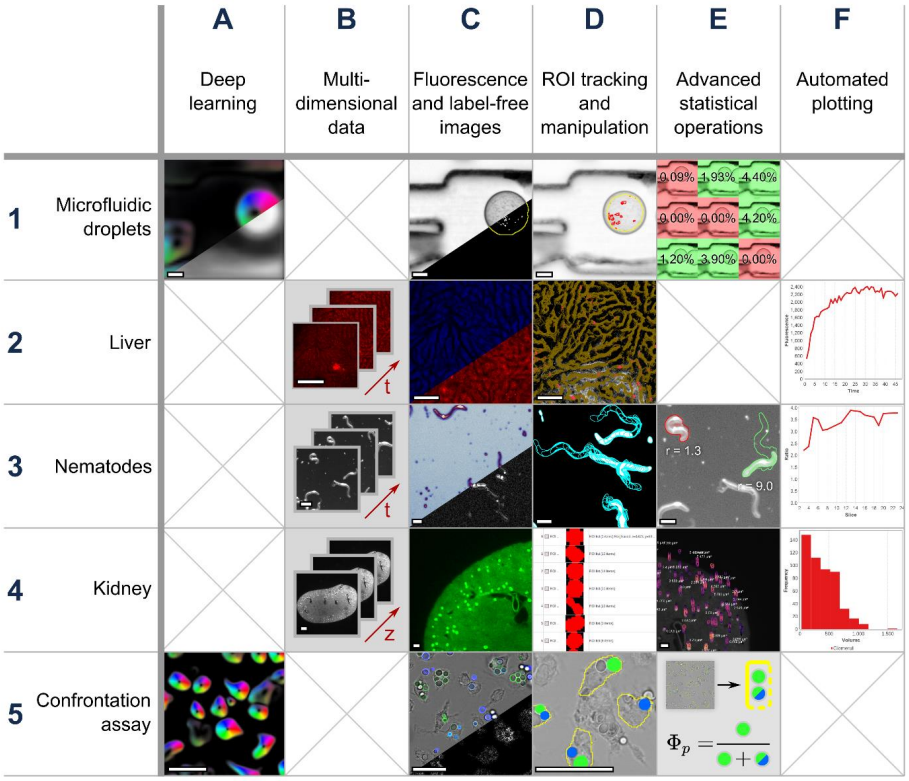


Figure 3 | Examples of biological applications and hallmarks of *JIPipe*. The rows are arranged according to the biological samples, whereas the columns correspond to sets of main features of the *JIPipe* platform. The highlighted techniques include deep learning (**column A**), the handling of multidimensional images (**column B**), the analysis of label-free and fluorescence microscopy data (**column C**), the flexible tools of region of interest (ROI) tracking and manipulations (**column D**), mathematical and statistical operations (**column E**), and data plotting via automatable nodes (**column F**). The rows show examples from the analysis of microfluidic droplets (**row 1**), intravital microscopy of the liver (**row 2**), time-tracking of live nematodes (**row 3**), the segmentation and quantification of kidney glomeruli (**row 4**), and the analysis of confrontation assays between macrophages and fungal spores (**row 5**). (**A-1**) The gradient flow (top left) and probability map (bottom right) of microfluidic droplets, provided by the *Cellpose* algorithm. (**A-5**) The gradient flow of unlabeled macrophages provided by the *Cellpose* node output. (**B-2**) Representative images from a time series of fluorescence images recorded from live liver. (**B3**) Earth worms in transmitted light microscopy images during a time series microscopy experiment. (**B-4**) Selected z slices of a 3D light-sheet microscopy image of mouse kidney. (**C-1**) Microfluidic droplets in a TL image (top left), as well as after segmentation of the droplet inner area (bottom right, yellow line) and of the bacteria (bottom right, white speckles). (**C-2**) Images of autofluorescence (top left, blue) and cargo-linked fluorescence labeling (bottom right, red) of a liver sample during intravital microscopy. (**C-3**) Nematodes in a live time-series TL microscopy experiment before processing (top left, "BuPu" look-up table[27]) and after Hessian filtering (bottom right). (**C-4**) One z slice of a 3D fluorescence microscopy image of mouse kidney (light green), overlaid with a maximum intensity z projection of the segmented glomeruli ROIs (bright green speckles). (**C-5**) TL images of a confrontation assay (top left) with the macrophages shown in grey, the phagocytosed spores shown in green, and the external conidia presented in blue. The macrophages were identified without fluorescence labeling by applying a Hessian filter (bottom right). (**D-1**) A microfluidic droplet imaged via TL microscopy, with the segmented inner line of the droplet shown in yellow, and the bacterial growth indicated by red. (**D-2**) Mouse liver as revealed by

intravital microscopy, showing the sinusoids as a time series maximum projection (dark yellow) superimposed with the time-series projection of LSECs (red speckles). (**D-3**) The time superposition of nematode outlines during a TL microscopy experiment (blue: nematode outlines at consecutive time points, white: binary image of segmented worms). (**D-4**) A selection of detected 3D ROIs representing individual glomeruli from a light-sheet microscopy experiment displayed in the JIPipe cache GUI. (**D-5**) The outline of segmented macrophages (yellow) superimposed with phagocytosed conidia (green) and adherent spores (blue). (**E-1**) Bacterial growth levels about 0.5% area/area were classified as positive growth (green hue), whereas lower values than 0.5% were considered as negative growth (red hue). The superimposed numbers indicate the per-droplet growth percentage values. (**E-3**) Nematodes imaged with TL microscopy, recorded as part of a time series. The red and green outlines are the footprints of two individual worms, one each with low and high motility values (red, motility ratio at 1.3; green, motility ratio at 9.0). (**E-4**) The volume values of individual glomeruli are calculated from light-sheet microscopy images of the mouse kidney. The original image appears as a greyscale background, the ROIs of the segmented glomeruli are shown in perspective view as generated by a special *JIPipe* node, where the individual 2D ROIs are colored according to their z position, and the volume values are printed in white. (**E-5**) Representation of the principle behind calculating the phagocytosis ratio from confrontation assay images. The segmented macrophages (yellow outline) were matched against the green and blue-labelled conidia in order to find the phagocytosed (green) and adherent (blue) spores. Using the *JIPipe* table calculator node, the ratio of the phagocytosed conidia over the total number of spores (phagocytosed plus adherent) was determined (Φ_p). (**F-2**) Using *JIPipe*'s automated plotting nodes, the mean fluorescence value of the cargo-linked dye measured in the hepatocytes was presented for a time series experiment (red curve). (**F-3**) The motility ratio of an individual worm was plotted over the course of a time series experiment (red curve). The ratio was defined as the total footprint of the worm covered during the video divided by the area of the worm at each time point. (**F-4**) Histogram representation of the glomeruli volume for a light-sheet microscopy image set measured in a mouse kidney. Scalebars represent 200 µm in column B, 100 µm elsewhere.

***JIPipe* user interface and data model.** Here we will provide a brief overview of some of the technical details of *JIPipe*, whereas the full organization of our software can be retrieved from the Supplementary Material (3: "JIPipe user interface and data model"), as well as from the *JIPipe* website (http://www.JIPipe.org/). Whilst *JIPipe* builds upon *ImageJ* functionality, it also contributes its own solution for handling (meta-)data and encapsulating it into a GUI. *JIPipe*'s backbone is a directed acyclic graph (DAG) that organizes all functional units, the so-called nodes. A pipeline is formed by connecting nodes through their input and output slots, thus modelling the flow of data. We simplified this concept by the introduction of compartmentalization, which clearly separates projects with a large number of nodes into functional units. *JIPipe* provides over 1000 nodes that cover every core aspect of image analysis and data management, and can be easily searched by name, functionality, and compatibility to the preceding node. To simplify the creation of workflows, many nodes were manually provided with integrated documentation and customization options, for example to test multiple parameter sets, or to apply calculations via user-defined mathematical expressions. These features are made possible by the table-based data management that enforces constraints, for example on data types, presence of a primary data column, or type of annotations. The consequence of a simplified table is zero-cost scaling, meaning that *JIPipe* workflows can be adapted to batch processing without any changes to the pipeline structure. At the same time, a table allows for great flexibility and can be utilized to track biological conditions, dataset identifiers, or image properties, thus providing an easy way to find and reproduce data and analysis details according to the FAIR principles[3]. According to the symbiotic relationship detailed earlier, all nodes and *JIPipe* algorithms can also be directly accessed from *ImageJ*. Pipelines can be executed in their entirety or up to a user-specified node, with the option of saving the results into a memory cache or on the hard-drive in a standardized format, including all parameters and the complete project file. This allows *JIPipe* to open existing results, making further processing and plotting at a later timepoint possible.

In conclusion, *JIPipe* is a user-friendly and powerful tool that introduces visual programming into *ImageJ*. Users who are already familiar with the *ImageJ* software find equivalent functionalities and can continue to use their existing scripts, in both *JIPipe* and *ImageJ*. Batch analyses can be designed efficiently by laying out the steps for a single analysis and then increasing the set of input files – all required functionality to

track datasets during the analysis are handled by the powerful table-based data architecture. We showed by the examples presented in **Fig. 3** that *JIPipe*, like *ImageJ*, can be applied to virtually all kind of biological data. *JIPipe* is fully open-source and licensed under BSD-2. Detailed user guides, tutorials, videos, examples, and instructions for developers are available on https://www.*JIPipe*.org/. We are working with the image analysis community (https://forum.image.sc/) to further improve *JIPipe* and establish the software as meaningful contribution to collaborative environments.

Discussion

With the advent of visual programming in many application fields, the need for a visual language for *ImageJ* programming has become apparent. This motivated us to develop *JIPipe* and by that fill a niche that bridges the gap between researchers with limited programming experience and the advancing development of modern methods of automated image processing, including deep learning approaches, parallel- and cloud computing. *ImageJ*[1] gained immense popularity[2] amongst biologists and bioimage informaticians due to its very wide applicability, ease of use and of expansion, and open-source nature. When it comes to analyzing large amounts of images, especially in a complex structure due, e.g., to a wide set of biological conditions, pipelines in *ImageJ* become complex, making it difficult to record the individual steps and their parameters according to the FAIR principles[3]. The necessity to have certain programming knowledge to turn single-image analysis workflows into complex batch analysis via macro or plugin programming discourages many less experienced users from developing automated pipelines in *ImageJ*. Here we identified the niche, where our contribution to visual programming can be meaningful for the image analysis community. With *JIPipe*, we established such a visual programming environment, where current *ImageJ* users can utilize their existing knowledge of *ImageJ* to convert existing pipelines into fully automated batch-processing workflows that generate exactly the same results as the *ImageJ* equivalents.

At this stage, *JIPipe* focuses on implementing the first version of the *ImageJ* API (*ImageJ1*)[28], because the majority of functions in the *ImageJ* GUI still rely on *ImageJ1*, where only one non-standardized string is available for transferring algorithm parameters. In addition, there is a concept of a single “active image”, a single table of results and ROI manager — restrictions that are not compatible with visual programming. In addition, custom GUI components are not suitable for GUI-free and parallel-execution environments. For this reason, we manually curated these functionalities into *JIPipe* nodes, instead of applying an automated algorithm for the conversion. Functions were added, merged, or split into multiple nodes, whereas the concepts of image windows, result tables and ROI manager were adapted into data types compatible with visual programming. As a result, *JIPipe* functions cover the same or very similar tasks as the corresponding commands in *ImageJ*. Due to this conversion process, it was not always sensible to keep the same function names as in *ImageJ*, adding a small learning curve to *JIPipe*. To ease this transition phase, we provide integrated search options, numerous application examples with complete code and data, as well as a thorough online training material (see **Supplementary Information 4**, https://www.JIPipe.org/tutorials/).

ImageJ incorporates a second, more modern API, *ImageJ2*, which is richer in features. For example, there are standardized interfaces for data and parameters, flexible service-based architecture, as well as individually addressable parameters. We adapted our approach of curating the integration via *ImageJ2* by incorporating the Multi-Template-Matching plugin[13]. An automated integration of all compatible operations was achieved by interfacing with the *ImageJ* Ops API[2] and applying on-demand conversion between *JIPipe* and *Image2* data.

In terms of documentations, *JIPipe* adopts the principle of immediate access to all functionality descriptions, greatly simplifying visual programming. The automated documentation tool was built from manually recovered online *ImageJ* documentation and source code, and it is available directly from the nodes.

Two of the most established and best-known visual applications in image analysis are *Icy* and *KNIME*. These tools integrate certain aspects of *ImageJ* into their own framework, while *ImageJ* cannot use functions designed for these tools, thus lacking the symbiotic relationship with *ImageJ*. This means that *ImageJ* itself is not the main focus of operation for these visual applications. Consequently, a niche was open for an application that would provide (i) a GUI for *ImageJ* operations with easy batch processing, automated saving, maintained full compatibility with *ImageJ*, (ii) a learning curve that makes this new language accessible for non-programmers, and (iii) a standardized project format that saves all information about the progress and implementation of the project according to the FAIR principles. This niche is filled by *JIPipe*.

Methods

Software development. *JIPipe* was developed in Java version 8 and utilizes software libraries provided by *ImageJ* and other developers. A full list of all dependency libraries is given in **Supplementary Table 5.1**, whereas the system requirements are listed in **Supplementary Information 5.2**.

Software availability. The version of *JIPipe*, and its source code used in this paper is available as supplement. The newest version of *JIPipe* and the current version of the source code are available on our website https://www.*JIPipe*.org/ and on https://www.github.com/applied-systems-biology/JIPipe/ (https://doi.org/10.5281/zenodo.6532719). *JIPipe* is also available from within the *ImageJ* update service. Instructions are available on our website.

Data availability. The data that supports our findings are available as supplements. This is also the case for the project files and example data used to demonstrate the usage of *JIPipe*. The example pipelines are available at https://doi.org/10.6084/m9.figshare.19733320.v1.

Acknowledgements

This work was financially supported by the International Leibniz Research School for Microbial and Biomolecular Interactions Jena – ILRS Jena. Furthermore, the German Research Foundation (DFG) funded this project through the Collaborative Research Center PolyTarget 1278 – project number 316213987, subproject Z01. This work was also supported by the Collaborative Research Center Funginet 124 – project number 210879364, subproject B4 – and by the Cluster of Excellence “Balance of the Microverse” under Germany´s Excellence Strategy – EXC 2051 – Project-ID 390713860, as well as by the Leibniz ScienceCampus *InfectoOptics* Jena, which is financed by the funding line Strategic Networking of the Leibniz Association. Furthermore, we received support from the Federal Ministry of Education and Research, Germany (grant number 13GW0456B) in the context of the *InfectoGnostics* Research Campus Jena. We are particularly thankful to Dr. Martin Roth (microfluidic droplets data), Prof. Christian Hertweck (nematode imaging), Drs. Kerstin Voigt and Mohamed Hassan (confrontation assay images), Dr. Adrian

Press (intravital liver microscopy data), and Prof. Matthias Gunzer (kidney light-sheet images) for kindly providing image data.

Author contributions

R.G. developed the software. Z.C. designed the bioimage analysis pipelines and tested the software. M.T.F. and Z.C. conceived the idea. M.T.F. directed and supervised the project. All authors wrote the initial draft, read and contributed to the paper and approved the content.

References

1. Rueden, C. T. *et al.* ImageJ2: ImageJ for the next generation of scientific image data. *BMC Bioinformatics* **18**, 529 (2017).
2. Schroeder, A. B. *et al.* The ImageJ ecosystem: Open-source software for image visualization, processing, and analysis. *Protein Sci.* **30**, 234–249 (2021).
3. Wilkinson, M. D. *et al.* The FAIR Guiding Principles for scientific data management and stewardship. *Sci. Data 2016 31* **3**, 1–9 (2016).
4. Berthold, M. R. *et al.* KNIME: The Konstanz Information Miner. in *Studies in Classification, Data Analysis, and Knowledge Organization (GfKL 2007)* (Springer, 2007).
5. de Chaumont, F. *et al.* Icy: an open bioimage informatics platform for extended reproducible research. *Nat. Methods* **9**, 690–696 (2012).
6. Haase, R. *et al.* CLIJ: GPU-accelerated image processing for everyone. *Nat. Methods* **17**, 5–6 (2020).
7. APEER. https://www.apeer.com/.
8. Afgan, E. *et al.* The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic Acids Res.* **46**, W537–W544 (2018).
9. Maloney, J., Resnick, M., Rusk, N., Silverman, B. & Eastmond, E. The Scratch Programming Language and Environment. *ACM Trans. Comput. Educ.* **10**, 16:1-16:15 (2010).
10. Martins, G. G. *et al.* Highlights from the 2016-2020 NEUBIAS training schools for Bioimage Analysts: a success story and key asset for analysts and life scientists. *F1000Research* **10**, (2021).
11. Legland, D., Arganda-Carreras, I. & Andrey, P. MorphoLibJ: Integrated library and plugins for mathematical morphology with ImageJ. *Bioinformatics* **32**, 3532–3534 (2016).

12. Meijering, E. FeatureJ. https://imagescience.org/meijering/software/featurej/.

13. Thomas, L. S. V. & Gehrig, J. Multi-template matching: a versatile tool for object-localization in microscopy images. *BMC Bioinformatics* **21**, 44 (2020).

14. Allan, C. *et al.* OMERO: flexible, model-driven data management for experimental biology. *Nat. Methods* **9**, 245–253 (2012).

15. Linkert, M. *et al.* Metadata matters: access to image data in the real world. *J. Cell Biol.* **189**, 777–782 (2010).

16. Stringer, C., Wang, T., Michaelos, M. & Pachitariu, M. Cellpose: a generalist algorithm for cellular segmentation. *Nat. Methods* **18**, 100–106 (2021).

17. Python. https://www.python.org/.

18. Jython. https://www.jython.org/.

19. R Core Team & others. R: A language and environment for statistical computing. (2013).

20. Büttner, H. *et al.* Bacterial endosymbionts protect beneficial soil fungus from nematode attack. *Proc. Natl. Acad. Sci. U. S. A.* **118**, 2110669118 (2021).

21. Hoffmann, B. *et al.* Spatial quantification of clinical biomarker pharmacokinetics through deep learning-based segmentation and signal-oriented analysis of MSOT data. *Photoacoustics* **26**, 100361 (2022).

22. JFreeChart. https://jfree.org/jfreechart/.

23. Svensson, C. M. *et al.* Coding of Experimental Conditions in Microfluidic Droplet Assays Using Colored Beads and Machine Learning Supported Image Analysis. *Small* **15**, 1802384 (2019).

24. Muljajew, I. *et al.* Stealth Effect of Short Polyoxazolines in Graft Copolymers: Minor Changes of Backbone End Group Determine Liver Cell-Type Specificity. *ACS Nano* **15**, 12298–12313 (2021).

25. Klingberg, A. *et al.* Fully Automated Evaluation of Total Glomerular Number and Capillary Tuft Size in Nephritic Kidneys Using Lightsheet Microscopy. *J. Am. Soc. Nephrol. JASN* **28**, 452–459 (2017).

408 26. Cseresnyes, Z., Kraibooj, K. & Figge, M. T. Hessian-based quantitative image analysis of host-
409 pathogen confrontation assays. *Cytometry A* **93**, 346–356 (2018).
410 27. Hunter, J. D. Matplotlib: A 2D graphics environment. *Comput. Sci. Eng.* **9**, 90–95 (2007).
411 28. Schneider, C. A., Rasband, W. S. & Eliceiri, K. W. NIH Image to ImageJ: 25 years of image analysis.
412 *Nat. Methods* **9**, 671–675 (2012).
413

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- lPipeNatureMethodsSupplementsfinal.pdf

Supplementary information

JIPipe: Visual batch processing for *ImageJ*

Ruman Gerst[1,2,#], Zoltán Cseresnyés[1,#], Marc Thilo Figge[1,3,*]

[1] Applied Systems Biology, Leibniz Institute for Natural Product Research and Infection Biology – Hans Knöll Institute (HKI)

[2] Faculty of Biological Sciences, Friedrich-Schiller-University Jena, Germany

[3] Institute of Microbiology, Faculty of Biological Sciences, Friedrich-Schiller-University Jena, Germany

[#] These authors contributed equally

[*] Correspondence should be addressed to thilo.figge@leibniz-hki.de

Contents

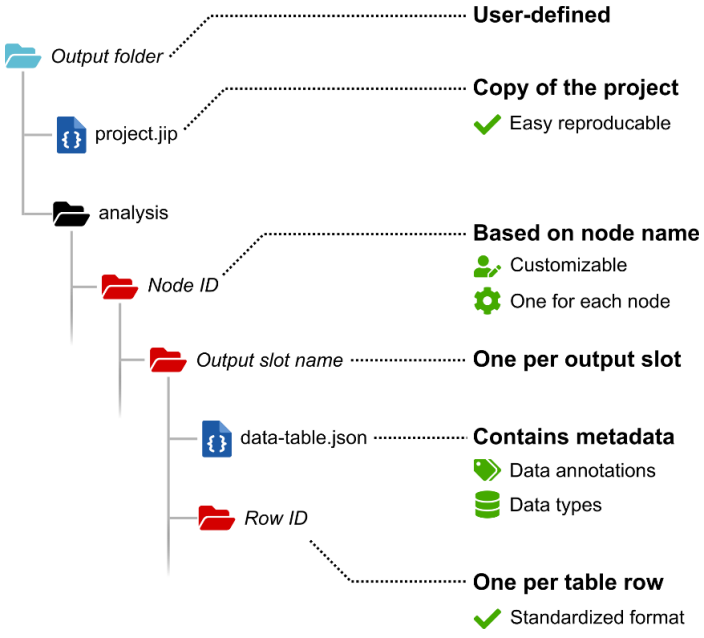
1 Symbiosis of *ImageJ* and *JIPipe*

1.1 Standardized output format

JIPipe writes results in a standardized format that allows results and annotations to be imported back into *JIPipe*. (see **Supplementary Figure 1.1**). The user only must provide the output folder. Here, *JIPipe* creates a sub-folder "analysis" that contains directories that correspond to the nodes in the graph. The name of these node folders is generated automatically and with de-duplication based on the user-customizable name of the node. This ensures that no data is overwritten, while users are still able to navigate through the results manually. Each node folder contains sub-directories that correspond to the output slots. *JIPipe* automatically ensures during the creation of these slots that they are compatible to filesystems and unique. Each of those slot folders contains metadata in a file "data-table.json". This file stores information about the data stored within the slot, annotations, expected data types, and true data types. Each row is also indexed with a unique identifier. Data is stored in sum-directories of the slot folder that correspond to this unique row ID. The format is defined by the stored data type and allows import back into *JIPipe*, as the metadata table provides all necessary info to direct *JIPipe* to the correct import routine. To improve the user experience, the metadata table is also present in CSV format that can be opened in standard software.

JIPipe makes use of this powerful result model by offering nodes that can import such results back into another analysis. The user only must provide the slot folder to allow *JIPipe* to import all data and metadata. This allows for postprocessing analyses that combine and structure results from multiple analyses.

To improve the usability of *JIPipe*, the standardized output format can be exported into a more commonly used where file names contain metadata. This feature is available within the cache browser, result viewer, and as dedicated node. This metadata-based export cannot be directly imported back into *JIPipe*.



Supplementary Figure 1.1 | Standardized result export format. *JIPipe* writes outputs in a standardized format. The user only must define the output directory (blue folder). *JIPipe* automatically generates a filesystem hierarchy based on the unique node Ids, output slot names, and row number in the output table (red folders).

1.2 Overview of *JIPipe* operations

JIPipe comes with a set of standard libraries that contain extensions for image analysis and other functions. It currently includes over 1000 nodes and over 120 data types. The function of the library components is briefly described here:

Filesystem library. A library of nodes that allow querying and manipulating file systems. It allows, for example, to search for files in a specified directory.

Annotation library. This library contains nodes that allow manipulation of data annotations.

Multi-parameter library. The core library provides the functions to execute a node on multiple parameter sets but lacks nodes to define such parameter sets. This library adds this functionality.

String library. A library that provides string data types (e.g., XML or JSON data).

***ImageJ* data type library.** This library integrates commonly used *ImageJ* functions, such as images, tables, and ROI management. Image data types are available in variants that restrict the bit depth or the dimensionality. Automated conversion is applied to ensure that the constraints are satisfied, for example, 8-bit grayscale images are converted into RGB images automatically. As each of these modes are available as separate data type, node developers and users can exactly control and review the inputs and outputs of a node, which improves usability and reduces the number of errors. This library also includes support for Bio-Formats[3].

***ImageJ1* algorithm library.** Commonly used commands from *ImageJ1* are integrated via this library. It provides functions to process images and ROI. The library also includes a macro node that can execute *ImageJ* macro code inside a *JIPipe* node.

***ImageJ2* algorithm library.** *ImageJ2* operations are automatically included via a translation layer.

CLIJ integration library. This library integrates functions from *CLIJ2*[1] into *JIPipe*. To improve performance, it provides a separate data type that encapsulates a GPU image and provides conversion from and to *ImageJ* images for ease of use. The functions were generated in an automated fashion via a Python script.

Table library. A library containing nodes that apply commonly used table operations (e.g., merging rows, or sorting). This library also adds data types that encapsulate only one table column for more advanced operations.

Forms library. Users can create interactive nodes that prompt the user to provide an input to the current data processing. There is an expandable set of such predefined input types available: Numeric inputs, check boxes, text fields, a choice of predefined values, and selecting a file system path. Multiple of these inputs can be connected into a form that is displayed for each processed data item. The standard forms library then writes user inputs into the annotations of the provided data. The library is modularized, which allows more complex form types, such as letting users draw or modify a mask interactively.

OMERO integration. *JIPipe* provides an integration into OMERO[2] that allows to query the database and download or upload images.

Python integration. *ImageJ* provides support for Python scripts via *Jython* (https://www.jython.org/) and a standard *Python* setup. The difference between *Jython* and Python is that *Jython* has access to all Java data types, including ones from *ImageJ* and *JIPipe* – while it is currently not possible to integrate C-based packages, such as *Numpy* or *Tensorflow*. To allow the integration of such powerful tools, *JIPipe* provides an environment system to integrate any existing Python environment. To increase usability

of this approach, *JIPipe* also comes with one-click installers to setup new *Python* environments via *Conda*.

Python scripts communicate with *JIPipe* via a file-based API. *JIPipe* automatically includes a *Python* library into Python scripts to make use of any node-specific functionality, such as accessing inputs and writing outputs.

R integration. *JIPipe* utilizes an environment system similar to *Python* environments to integrate *R* scripts. Similar to *Python*, users will find nodes to integrate custom *R* scripts into the pipeline. Again, *JIPipe* provides a file-based API to communicate data and metadata with the R script.

Cellpose integration. We included the ability to run *Cellpose* into *JIPipe*. As *Cellpose* is a *Python*-based tool, we make use of the *Python* integration functionality. *JIPipe* supports segmentation with *Cellpose* on the provided pretrained cytoplasm/nuclei models or a custom model. We also included the ability to train new models – either from scratch or by retraining a custom or pretrained model.

Utility library. Miscellaneous functions, like interaction with *JIPipe* outputs, manual data conversion, and data table sorting.

1.3 Extension API

All non-core functionality is split into dedicated Java libraries that are usually distributed with the *core* library but can be left out for specialized distributions of *JIPipe* that focus on other usages like non-image-analysis workflows. If only the *core* library is loaded, *JIPipe* will contain no usable data types and nodes. Image analysis functions are provided in dedicated libraries as extensions. Extensions for *JIPipe* are *SciJava* plugins that provide the necessary metadata for *JIPipe*, for example the name, authors, and dependencies, and a `register()` function that executes all necessary steps to add additional functionalities. Following functions can be added via this function:

Data types. Java developers can add new data types into *JIPipe*. Data is organized into tables and annotated with additional string columns. To allow for automated reading and saving, data types must provide functions to export itself into a folder and be imported from an exported directory. Additionally, data types must be able to be displayed in the GUI, via a `display()` function and an optional preview method. Each data type has a unique identifier string that allows safe serialization of user-customizable slot configurations.

Node types. Node types are Java classes that contain the workload function. Slots are either added via Java annotations or created in the object constructor. Like data types, they have a unique identifier.

Data type conversions. *JIPipe* automatically applies trivial conversions (e.g., from a child class to one of its parent classes). Other conversions (e.g., converting a plot to an image) must be handled by a dedicated converter object that can be registered into *JIPipe*. The converter creates an edge within the conversion graph, which allows higher-order conversions with multiple steps.

Data type display operation. Each data object comes with a default function to display the data in the GUI (e.g., displaying an image in *ImageJ*). Additional display operations can be registered via an extension.

Data type import operation. An import operation imports data from a *JIPipe* result folder and displays it to the user. An example is the import and display of ROI.

Parameter type. Developers can register custom parameter types. Each parameter type requires a unique identifier, required for serialization of user-defined parameters, and a user interface.

Expression function. The set of functions available in expression parameters can be further expanded.

Table column operation. Independent of expression functions, there exist functions that apply one-to-one or integrating operations on table columns. This set can be expanded. Such operations are automatically available inside expressions.

Menu extensions. Developers can create custom menu entries for various uses. They can choose between multiple locations (e.g., "Project" menu or "Tools" menu).

An alternative to Java extensions is extensions provided as JSON files. They can be created via a user-friendly GUI from existing pipelines or sets of nodes and allows non-developers to create custom node types, akin to *ImageJ2* scripts or macros. Such extensions are loaded via a "JSON Extension Loader" Java extension that automatically scans the *ImageJ* plugins directory for valid extensions.

2 Hallmarks of *JIPipe* by representative applications

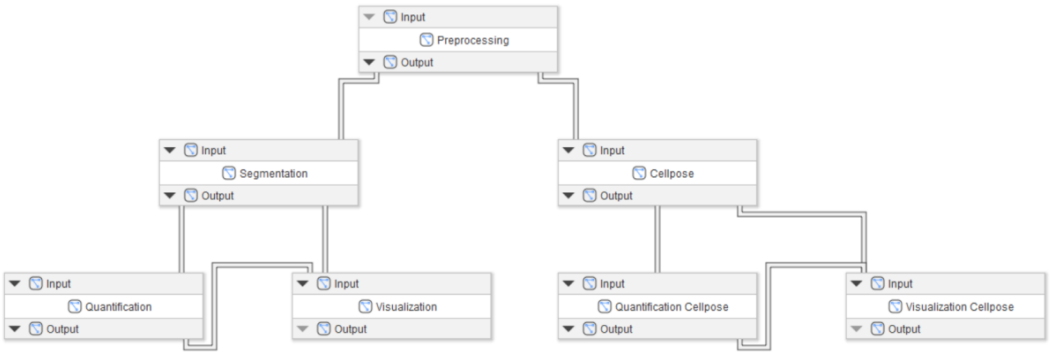
Here we describe the details of the *JIPipe* application that were utilized to illustrate the wide applicability of *JIPipe* in the Results. The examples are the following:

a) Bacterial growth inside fluid droplets
b) Nanoparticle delivery in liver
c) Host-pathogen interactions
d) Nematode viability test
e) Kidney status check via glomeruli counting

2.1 Bacterial growth measured in fluid droplets

Picoliter droplets are miniature bioreactors used in microfluidic experiments to test various growth conditions on bacteria[3–5]. Due to the extremely large number of droplet images, the native batch-processing and parallelization features made *JIPipe* an ideal candidate to quantify the bacterial growth in potentially millions of droplets[3] (**Supplementary Figure 2.1**). The workflow was successful in finding the targeted droplet, identify its inner zone and detect bacterial growth (**Figure 3 Row 1**). In this case, microfluidic droplets of approximately 100 micrometer diameter were filled with a solution containing *E. coli* bacteria and the bacterial growth was observed via brightfield transmitted light microscopy[3]. The following *JIPipe* workflow determines the droplets that show bacterial growth. When compared with a set of 1500 images with manual annotations (growth vs. no growth), the *JIPipe* workflow produced 100% agreement with the ground truth.

The processing workflow starts with scanning the input file folder(s) and annotating the internal *JIPipe* table with the folder names and the image identifiers. It is sufficient to drop only the top folder into the flow (node "Folder list") because the subsequent nodes will automatically extract the rest of the information, e.-g. the subfolders (here we use the Recursive list option in the Parameters setting to handle multiple layers of subfolders automatically). In the "List files" node we search for files that are of the CZI type by introducing a filter for the absolute path. After adding the image names to the annotation table, we provide another filtering opportunity by the "Filter paths" node, which can be useful when limiting the analysis to a subset of images during testing the analysis workflow.



Supplementary Figure 2.1 | Compartment graph of the classical and Cellpose-based image analysis approach to identify microfluidic droplets that show bacterial growth inside. For the node arrangement within individual compartments, see the supplied JIP project file "Droplets.jip" and the detailed nodes map “compartments-figure-droplets.png” in Supplementary Materials.

In detail:

(i) Preprocessing:
- Read the images into memory and annotate the data table as described in **Supplementary Information 2.1**
- Pass the results to the Segmentation and Cellpose compartments

(ii) Segmentation:
- Hessian segmentation(parameters 2, “Largest”, 2)
- Gaussian blur (2 px)
- Auto threshold (Triangle algorithm)
- Morphological hole filling
- Distance transform watershed
- Morphological opening (10 px)
- Morphological erosion (7 px)
- Create ROI and split multiple droplets to create the outer line
- Morphological erosion (5 px)
- Create ROI and split multiple droplets to create the inner line
- Pass the results to the Quantification compartment

(iii) Quantification:
- Filter ROI by statistics (Area above 1000)
- Calculate variance (1 px)
- Auto threshold (Otsu algorithm)
- Extract ROI measurements (Area, Area fraction, Integrated density)
- Create Growth column (percentage Area above 0.5 is classified as growth = 1)
- Pass the results to the Visualization compartment

(iv) Visualization:
- Convert growth areas into ROIs
- Merge raw image with growth area ROIs
- Convert inner line of droplets into ROIs
- Merge raw image with inner line ROIs

(v) Cellpose:

- Access the TL images from Preprocessing
- Run Cellpose segmentation
  - Object diameter 120 px
  - Use pretrained Cellpose model “Cytoplasm”
  - Thresholds: 2 (probability), 0.8 (flow)
- Filter ROIs by Roundness > 0.7
- Morphological opening (10 px)
- Morphological erosion (1 px for outer line, 7 px for inner line of droplets)
- Turn masks to ROIs and split them

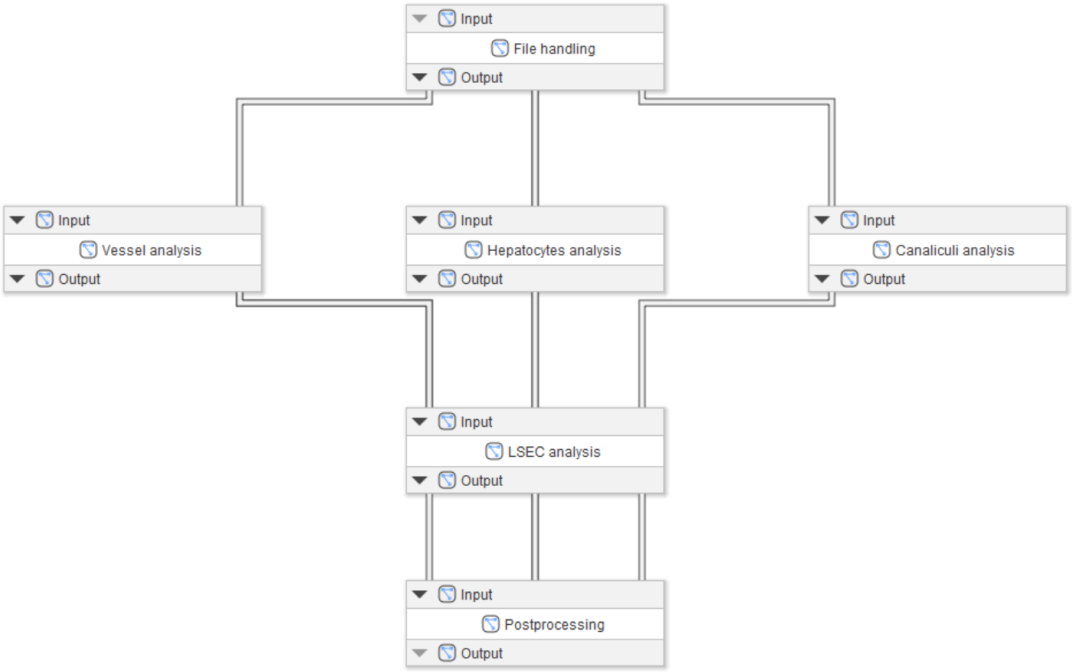
(vi) Quantification Cellpose: see Quantification

(vii) Visualization Cellpose: see Visualization

2.2 Nanoparticle delivery analysis in liver

JIPipe’s native batch processing ability and built-in time series algorithms were of particular advantage in a project using nanoparticles (NPs) to counteract liver fibrosis caused by non-alcoholic fatty liver disease. NPs are utilized to deliver precisely targeted agents to the liver tissue[6]. The analysis of such microscopy data requires the identification of various liver components including hepatocytes, liver sinusoidal endothelial cells (LSECs), sinusoids, and canaliculi. The segmentation was carried out without the help of specific labeling, and the extraction of time series information about the uptake and extrusion of the NP-delivered agents. The extensive set of morphological filters available in *JIPipe* were invaluable in identifying the various components of the liver without specific labelling, based solely upon the autofluorescence signal (**Supplementary Figure 2.2)**. The resulting high-fidelity segmentation of the LSECs, canaliculi and sinusoids (**Figure 3 D-2**) indicate the precision and utility of the *JIPipe* processing framework.

The live-animal microscopy experiments were described in Muljajew et. al., 2021[7]. Micelle nanocarriers were injected into the circulatory system of the mouse vie the tail veins. Two-photon microscopy was utilized to image the cargo delivered by the micelles to the hepatocytes, sinusoids, canaliculi and liver-sinusoidal endothelial cells the time-series images were analyzed by the JIP protocol “LiverAnalysis.jip” (see Supplementary Materials)



Supplementary Figure 2.2 | Compartment graph of the analysis protocol for liver drug delivery assay, designed to quantify the spatio-temporal distribution of nanoparticle-delivered cargo to various parts of the murine liver. For the node arrangement within individual compartments, see the supplied JIP project file " LiverAnalysis.jip" and the detailed nodes map "compartments-figure-liver.png" in Supplementary Materials.

The workflow was based on principles similar to those shown in the previous chapter. The processing consisted of six compartments: i) file handling, ii) blood vessel analysis, iii) hepatocyte analysis, iv) canaliculi analysis, v) the analysis of liver sinusoidal endothelial cells (LSECs), and vi) postprocessing.

In detail:

(i) Images were read into memory and the data table was annotated as described in **Supplementary Information 2.1**

(ii) Vessel analysis:
- Median blur (radius=5 pixels)
- Illumination correction (20 px)
- Auto threshold (Yen algorithm)
- Despeckle
- Morphological erosion (7 px)
- Particle finder (circularity range 0.0-0.4)
- Multi-node algorithm to arrange the time series based on slice numbers
- Calculate the intensity time series for the segmented vessel region
- Calculate the number of segmented objects per time point to check the segmentation
- Pass the results to the LSEC analysis compartment

(iii) Hepatocyte analysis:
- Median blur (radius=5 pixels)
- Illumination correction (20 px)
- Auto threshold (Li algorithm)
- Despeckle

- Morphological erosion (1 px)
- Morphological skeletonize
- Particle finder (size range 10^2-10^6)
- Multi-node algorithm to arrange the time series based on slice numbers
- Calculate the intensity time series for the segmented vessel region
- Calculate the number of segmented objects per time point to check the segmentation

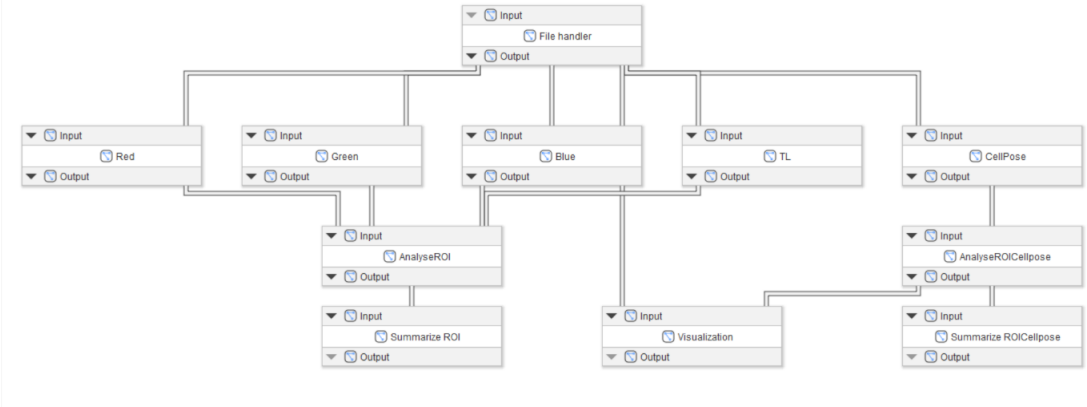
(iv) Canaliculi analysis
- Median blur (radius=5 pixels)
- Illumination correction (20 px)
- Auto threshold (Li algorithm)
- Despeckle
- Morphological erosion (1 px)
- Skeletonize
- Particle finder (no filtering)
- Multi-node algorithm to arrange the time series based on slice numbers
- Calculate the intensity time series for the segmented vessel region
- Calculate the number of segmented objects per time point to check the segmentation

(v) LSEC analysis:
- Take inputs from vessel analysis and file handler (fluorescence image)
- Illumination correction of fluorescence image (20 px)
- Create mask from segmented vessel image
- Morphological dilation (7 px)
- Mask fluorescence image with segmented vessel image
- Auto threshold (RenyiEntropy algorithm)
- Particle finder (no filtering)
- Multi-node algorithm to arrange the time series based on slice numbers
- Calculate the intensity time series for the segmented vessel region
- Calculate the number of segmented objects per time point to check the segmentation

2.3 Confrontation assays

The interaction between alveolar macrophages and fungal spores was examined as described in earlier research[6,8,9]. The macrophages and fungal spores were identified by label-free segmentation algorithms, whereas counterstained fungi were identified by fluorescence labeling[8–10]. The workflow was parallelized to segment labeled and unlabeled cells and spores separately. In addition, classical and deep learning—based approaches of the macrophage segmentation algorithms were also organized into separate parallel compartment groups (**Supplementary Figure 2.3**). The essential nodes consisted of Hessian filtering to identify unlabeled macrophages and fungal spores, background correction with appropriate parameters, thresholding steps, and fine-tuned morphological operators. Here the "Define multiple parameters" node was of high importance by allowing to test many parameters in one run. This special node allows the definition of one or more parameters that will be chosen to fit a processing node, with each parameter allowed to be given any number of values to be tested, and then connected to the corresponding processing node. For example, when testing various thresholding methods, a “Define multiple parameters” node was set up to contain the parameter "Method" with values set to the seventeen methods provided by *ImageJ*. The node was then plugged into an "Auto threshold 2D " node to provide an overview of the effectiveness of all seventeen methods on the test images in just one process. The outcome of the analysis consists of phagocytic measures

and of segmented images of all participants (host cells, phagocytosed, adherent, and free pathogens, phagocytosing, and passive macrophages), see **Figure 3 D-5**. For the classification of the segmented objects, ROI-analysis nodes were developed; these enable the quantification of ROI overlap, e.g., between host cells and fungi to identify phagocytosed spores. A set of the ROI comparison nodes were arranged into a separate compartment "Analyze ROI", followed by a set of nodes to calculate the various phagocytosis measures arranged in the compartment "Summarize ROIs". For the deep learning—based segmentation of the macrophages, the recently published Cellpose[9] method was fully integrated into *JIPipe*. The Cellpose-related nodes include "Cellpose" (to apply the Cellpose model either in its original form, or after transfer learning, or following training from scratch); "Cellpose training" (for transfer learning and training a model from the beginning); "Import Cellpose model" and "Import Cellpose size model" to read in an already trained model for predefined size or for trained object size, respectively. Using Cellpose via these *JIPipe* nodes vastly simplifies the workflow building process, which is of great advantage for those with little experience in applying Deep Learning methods in image analysis.



Supplementary Figure 2.3 | Compartment graph of the confrontation assay analysis protocol, designed to quantify host-pathogen interactions. For the node arrangement within individual compartments, see the supplied JIP project file " ConfrontationAssay.jip" and the detailed nodes map “compartments-figure-confrontation.png” in Supplementary Materials.

In the example provided here, we limited the analysis to the “LabeledHosts_LabeledPathogens” dataset, which contained images where both the immune cells (hosts) and the fungal spores (pathogens) were imaged not only in transmitted light modality, but also with fluorescence microscopy using specific labeling of the assay components. The images are then read into memory with the “Import image” node, and the channels are separated before passing the data into the output node. As shown in **Supplementary Figure 2.3**, the output node is connected to the subsequent five segmentation compartments: i) antibody-labeled hosts (“Red”), ii) FITC-labeled pathogens (“Green”), iii) calcofluor white (CFW)-labeled pathogens (“Blue”), iv) transmitted light images (“TL”), and v) the deep-learning based segmentation workflow (“CellPose”).

In detail:

(i) Images of the labeled host cells are processed as follows:
- Gaussian blur (radius=3 pixels)
- Internal gradient (25 px)
- Contrast enhancement
- Background subtraction (Rolling Ball, 50 px)
- Auto threshold (Triangle algorithm)
- Morphological closing (2 px)
- Morphological hole filling

- Watershed transformation
- Morphological erosion (5 px)
- Particle finder to identify macrophages by size (3000-30000) and circularity (0.1-1.0)

"Define multiple parameters" nodes were used originally to test a range of rolling ball radii, and a series of automated thresholding algorithms, respectively.

(ii) Images of the FITC-labeled fungi are processed as follows:
- Remove outliers (radius=20 pixels, threshold=5)
- Background subtraction (Rolling Ball, 22 px)
- Auto threshold (Triangle algorithm)
- Watershed transformation
- Particle finder to identify fungal spores by size (100-3000) and circularity (0.4-1.0)

Two "Define multiple parameters" nodes were used originally to test a range of rolling ball radii, and a series of automated thresholding algorithms, respectively.

(iii) Images of the CFW-labeled fungal cells are processed as follows:
- Remove outliers (radius=20 pixels, threshold=5)
- Contrast enhancement
- Background subtraction (Rolling Ball, 22 px)
- Auto threshold (Li algorithm)
- Watershed transformation
- Particle finder to identify fungal spores by size (100-3000) and circularity (0.4-1.0)

Two "Define multiple parameters" nodes were used originally to test a range of rolling ball radii, and a series of automated thresholding algorithms, respectively.

(iv) Images of the TL images of hosts and pathogens are processed as follows:
- Laplacian sharpening with a 3x3 kernel
- Hessian filtering using the smallest eigenvalues with smoothing of 3 pixels
- Gaussian blur (radius=5 px)
- Auto threshold (Huang algorithm)
- Annotating with maximum and minimum threshold values
- Morphological closing (2 px)
- Morphological hole filling
- Remove outliers (radius=10 pixels, threshold=20)
- Remove outliers (radius=20 pixels, threshold=20)
- Watershed transformation
- Morphological erosion (2 px)
- Particle finder to identify macrophages by size (3000-30000) and circularity (0.1-1.0)

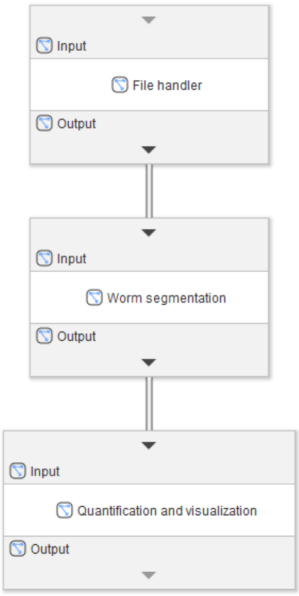
(v) The TL and fluorescence images of hosts and pathogens were also segmented using the default trained networks of the Cellpose environment[15]. Here no pre- or post-processing steps were applied. Rather, the outcome from the Cellpose node provided the ROI lists of the hosts and pathogens, and the lists were passed on to the output node, from where they were directed to the "AnalyseROICellpose" compartment (see below).

The segmented images are used to generate lists of regions of interest (ROIs) that describe the locations of the host cells, as well as the green-labeled and blue-labeled pathogens. These ROIs are further examined in the "AnalyseROI" and "AnalyseROICellpose" compartments (the latter one applied for the Cellpose-based analysis) via testing the overlap between pairs of the three object groups to identify associated fungal spores (i.e., fungi that are interacting with a host cell based on their

overlapping ROIs), adherent fungi (associated pathogens that are CFW-positive) and phagocytosed fungi (associated fungi that are not adherent, i.e. CFW-negative). In addition, phagocytosing host cells are identified as objects that contain at least one phagocytosed pathogen. In the last step, the "SummarizeROI" or "SummarizeROICellpose" compartments calculate the four phagocytic measures[12], using the "Modify tables" node that contain the calculations in a Python script.

2.4 Track analysis of unlabeled nematodes

When beneficial soil fungi are consumed by nematodes (earth worms), a way to protect the soil quality is to provide the fungi with symbiotic bacteria that produce agents that are toxic for the worms but not for the fungi, thus protecting the soil-enhancing fungi from the nematodes[11]. *JIPipe* was used to segment and track the nematodes, and to calculate a viability ratio (the total footprint area covered by a worm divided by the area of the worm averaged over time) that characterized the efficiency of various symbiotic bacteria to protect the fungi (**Supplementary Figure 2.4**). *JIPipe* was extended for this project with a node to find connected components that allowed the analysis of time series experiments. The outcome produced by the project included the merged outlines of every worm (**Figure 3 D-3)**, the binarized nematodes and the outline of one animal at a selected number of time points, as well as the footprint of a single worm superimposed onto the original images (**Figure 3 E-3)**. The postprocessing steps included the calculation of the worm areas and footprints (i.e., the collection of pixels that were touched by the worm during the course of the time series), and measuring the ratio between the footprint and the individual worm area, which measures the motility of the animal; the higher the ratio, the more motile the worm. The time-series images were analyzed by the JIP protocol "Nematodes.jip" (see Supplementary Materials).



Supplementary Figure 2.4 | Compartment graph of the kinetic analysis workflow designed to characterize nematodes according to their motility. For the node arrangement within individual compartments, see the supplied JIP project file "Nematodes.jip" and the detailed nodes map "compartments-figure-nematodes.png" in Supplementary Materials.

The workflow was based on principles similar to those shown in the previous chapters. The processing consisted of three compartments: i) file handling, ii) worm segmentation, iii) quantification and visualization.

In detail:

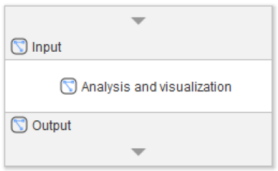
(i) Images were read into memory and the data table was annotated as described in **Supplementary Information 2.1**

(ii) Worm segmentation:
- Splitting stacks to reduce data size for easier testing (optional)
- Gaussian blur (3 px)
- Auto threshold (Triangle algorithm)
- Morphological closing (7 px, diamond)
- Morphological hole filling
- Particle finder (minimum size 4000 px)
- Pass the results to the Quantification and visualization compartment

(iii) Quantification and visualization:
- Time tracking individual worms based on the "#Component" annotation
- Create and measure worm area with logical OR
- Create total area per worm using the "Total" annotation
- Calculate total area over individual worm area
- Recreate time series using the "Slice" annotation in ascending order
- Calculate average, standard deviation and count of area ratios

2.5 Kidney status check via glomeruli counting

Kidney diseases, e.g. nephrotoxic nephritis lead to a diminished function of the kidney tissue, indicated by the reduced number of glomeruli[12]. Light sheet microscopy can be utilized to image whole kidneys in 3D. These images were generated by staining the glomeruli, the functional units of the kidney. Due to the high dimensionality of the data and the number of glomeruli that can range up to 16000, manual counting is highly time-consuming and impractical. Therefore, our group already developed fully automated solutions in Python and C++[13]. The disadvantage of these tools is that they require programming to be adapted and improved. Here, we exemplify how *JIPipe* can be used to apply an equivalent analysis, but without the need for programming (**Supplementary Figure 2.5**). For this example, we reduced the size of the image stack from 700 to 20, which even non-workstation computers can process without computing and memory capacity problems. The outcome of the *JIPipe* analysis included the identification of individual glomeruli (**Figure 3 E-4**) and the outline of the entire kidney tissue. We provide the *JIPipe* protocol file, as well as the input data in the Supplementary Materials. Here we also demonstrate the use of a single-compartment configuration of a *JIPipe* workflow, combing all processing and visualization steps into a single space.



Supplementary Figure 2.5 | Compartment graph of the glomeruli analysis workflow designed kidney light sheet microscopy images. For the node arrangement within individual compartments, see the supplied JIP project file "kidney_example_pipeline.jip" and the detailed nodes map "compartments-figure-glomeruli.png" in Supplementary Materials.

The processing workflow is organized into three logical steps: i) file handling: these are nodes for reading and organizing the input images, which are then passed on to the processing nodes

ii) glomeruli segmentation nodes; iii) tissue segmentation and quantification nodes; iv) quantification of glomeruli; v) visualization nodes: the segmented ROIs are quantified and plots are generated. These plots include glomerular number bar diagrams and a histogram of the glomerular volumes per kidney. The final plots of the tissue and glomeruli outlines can also be directly accessed via bookmarks. To execute or visit the bookmarked nodes, go to *Project → Project overview* and find the bookmarks on the right side in the "Bookmarks" tab. Click on any bookmark and choose either "Run" (to execute the pipeline up to that node, inclusive), or "Go to bookmark" to visit the node directly.

In detail:

(i) File handling:
- Images are provided as list of folders, containing the slices of an image stack. *JIPipe* converts these user-provided folders into a managed path data structure.
- Folders are annotated with their name that will be used in the pipeline to distinguish images from each other
- An "Import image stack" node is used to load the slices contained inside each folder into a 3D image. Annotations are preserved.
- The imported images are passed to the segmentation nodes.

(ii) Glomeruli segmentation:
- Input images are received from the output of the file handling nodes
- White Top Hat (radius = 5, disk shape) is applied
- Auto threshold (Otsu method)
- Morphological opening (radius = 2, disk shape)
- "Find Particles 2D" (default settings)
- "Split into connected components" is applied to the set of 2D ROIs generated by the particle finder. This node applies a 3D connected components algorithm and groups 2D ROIs of the same component into dedicated ROI lists. Each output ROI list is annotated with an identifier and corresponds to one glomerulus.
- The glomeruli are passed to the quantification and visualization nodes

(iii) Tissue segmentation and quantification:
- Input images are received from the output of the file handling nodes
- Median blur (radius = 1) is applied
- Auto threshold (Default method)
- Morphological closing (radius = 20, disk shape)
- Morphological hole filling
- Find particles (default settings)

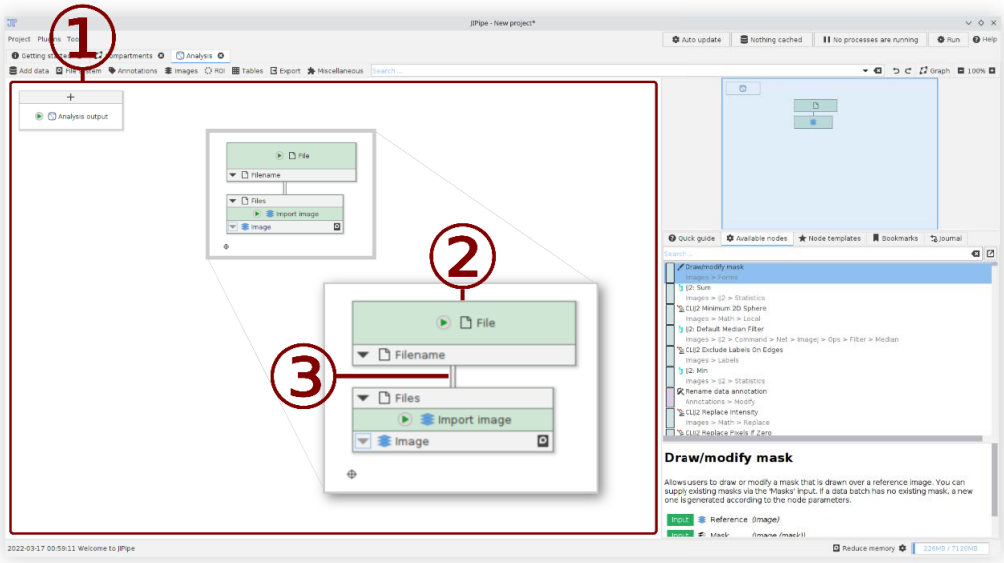
(iv) Quantification of glomeruli:
- "Extract ROI statistics" (Extracted measurements = "Area") creates a table with one row per 2D ROI containing its area
- "Integrate table columns" (Input column = Area, Function = Sum, Output column = Volume) calculates the volume in px^3 for each glomerulus. Its output is a table with one row
- "Add annotations as columns" (Annotation name filter: value == "#Component") adds the glomerulus identifier into each table
- "Merge table rows" (Data batches/Grouping method = "Custom", Data batches/Custom grouping columns = "#Dataset") merges all quantified results of the same kidney into one table
- "Filter table" (Volume >= 28 AND Volume <= 2300) removes glomeruli outside the expected volume range

(v) Visualization

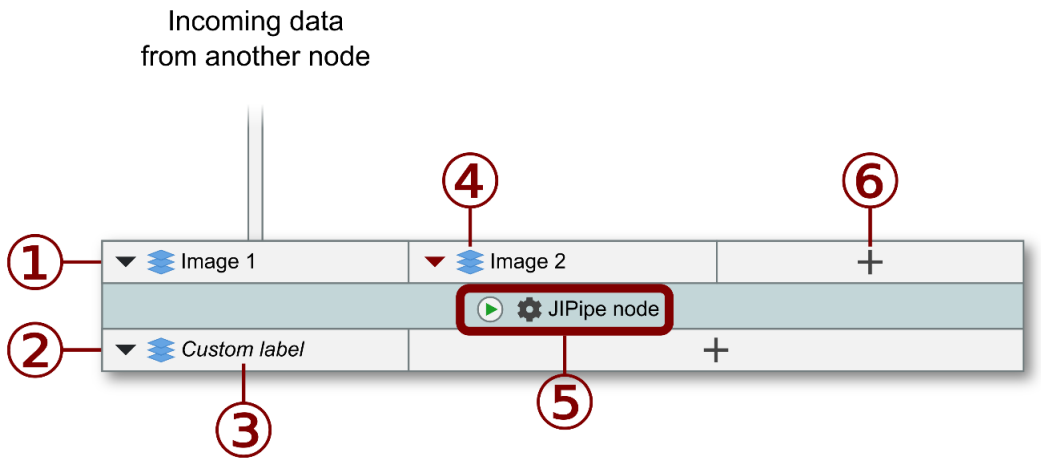
- The distribution of glomerular volumes is plotted via “Plot tables” (Plot type = Histogram, Value = Volume) plots the “Volume” column as histogram
- The tissue is visualized via a “Convert ROI to RGB” node that consumes the extracted tissue ROI and the raw input image enhanced via a “Histogram-based contrast enhancer”. A “Change ROI properties” node modifies the ROI to be drawn in green
- A combined visualization is generated as follows:
- The glomerular ROIs are modified via “Change ROI properties” to be drawn as yellow lines
- “Merge ROI lists” combines the glomerular ROIs and the tissue ROI into a single list
- A “Convert ROI to RGB” node overlays the glomeruli and tissue ROIs on top of the tissue

3 *JIPipe* user interface and data model

Here we focus on key features of the *JIPipe* user interface and explain how our software implements a scalable data model. The full organization of our software can be retrieved from the Supplementary Material as well as from the *JIPipe* website (http://www.JIPipe.org/). Familiarizing with the user interface is assisted by numerous training videos that can be accessed at the website as well. The central component of the *JIPipe* GUI is a graph that contains all functional units in form of nodes (see **Supplementary Figure 3.1**). Each of these nodes has one or multiple input and output slots that represent the data entered and produced by this functional unit (see **Supplementary Figure 3.2**). To create a pipeline, these slots are connected via edges to indicate a transfer of data from one node’s output to another node’s input. Outputs can be connected to multiple inputs, e.g., for creating branches to apply different methods or to generate visualizations of intermediate steps.

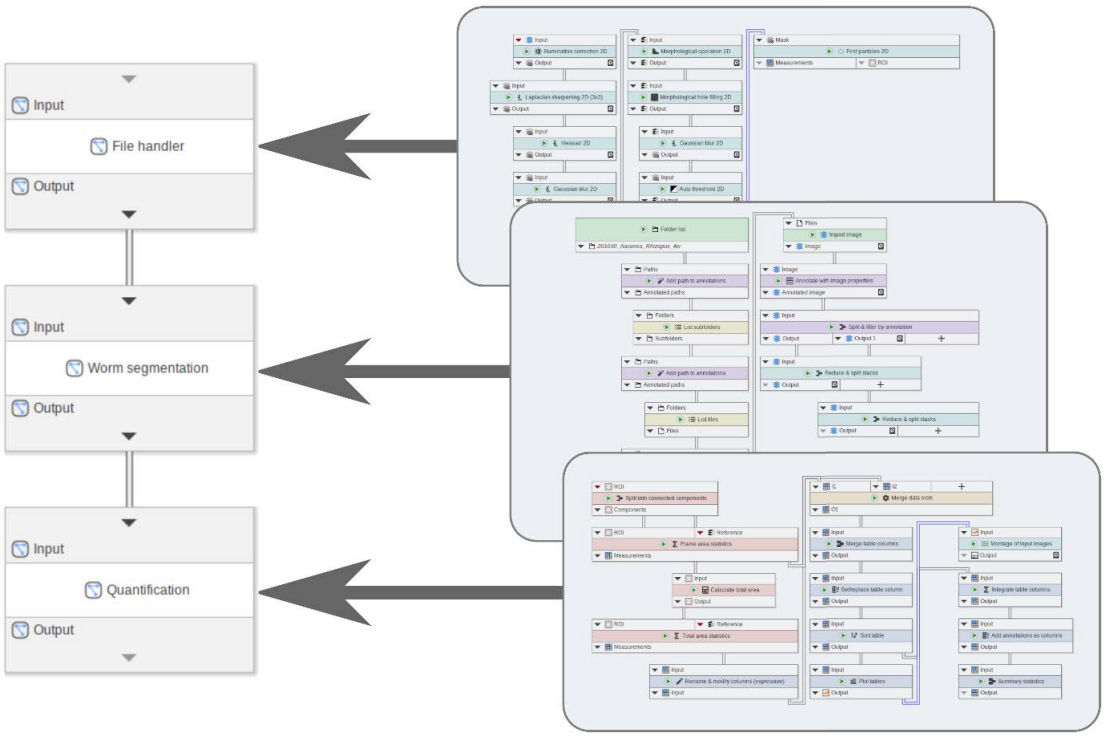


Supplementary Figure 3.1 | *JIPipe* graph editor UI. ① The central graph area where operational nodes can be placed by the user. ② Graphical representation of a node in *JIPipe*. Nodes have one or multiple input and output slots (grey areas within each node). ③ Output slots can be connected to inputs via edges (gray line).



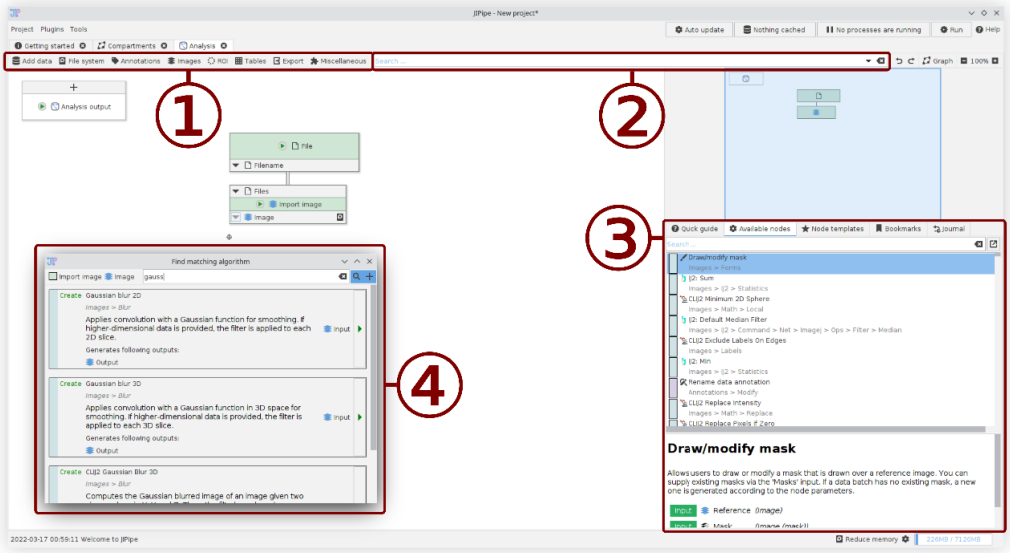
Supplementary Figure 3.2 | Graphical representation of a node with two inputs and one output slot. Only one input is connected (grey line). ① Inputs of the node are in the top row. ② The bottom row contains the node's outputs. ③ Users can customize the label names. These are displayed in an italic style. ④ Each slot displays its supported data type as icon. ⑤ The middle row contains the customizable node name, its icon representation, and a button to run the node and its predecessors. ⑥ Various nodes allow the creation of custom slots by clicking the "+" button.

Users are able to freely compartmentalize their pipelines (see **Supplementary Figure 3.3**), for example into preprocessing, segmentation, and postprocessing (see **Fig. 2a**). Within compartments, users can add additional nodes via a menu and arrange them freely.



Supplementary Figure 3.3 | Compartmentalization of pipelines. Users can use a compartment graph to organize a pipeline. Compartments are created and connected via a dedicated compartment graph (left). Each node in this structure contains a space where functional nodes can be placed (right).

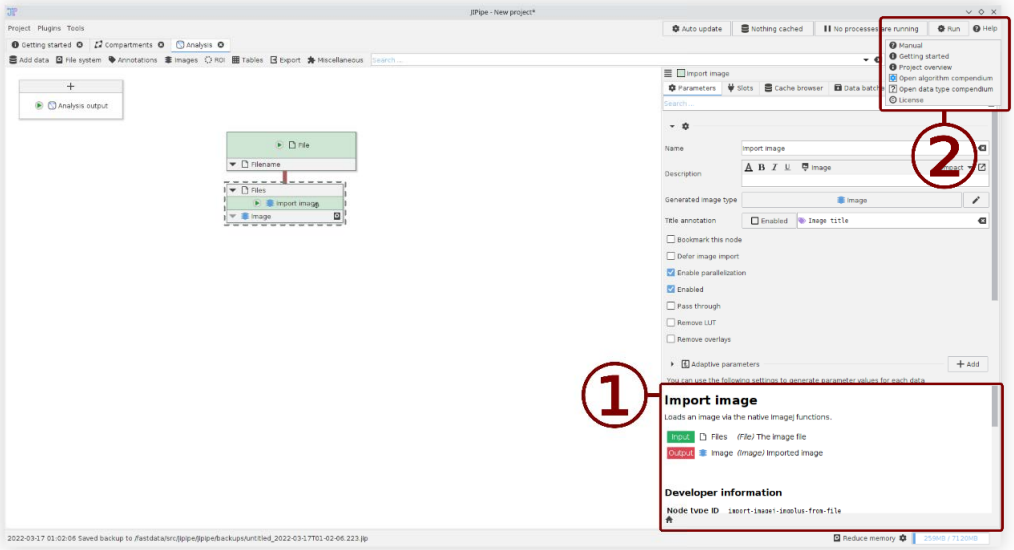
JIPipe provides over 1000 nodes that include tools for data management and generation; mapping the file and folder structure of the data; annotation tools to keep track of file origins and experimental conditions; image processing nodes; ROI management; table processing and filtering; nodes to export the results in various formats and structures; and a group of additional miscellaneous nodes to add utilities to the system. Multiple ways are provided to search for specific nodes (see **Supplementary Figure 3.4**), e.g., by name, functionality, and compatibility to the preceding node. According to the symbiotic principle outlined earlier, these nodes can also be directly accessed from *ImageJ*.



Supplementary Figure 3.4 | GUI functions to add nodes into a pipeline. ① Nodes are organized into a menu. ② New and existing nodes can be searched via a search bar. ③ Users who are familiar with other visual programming languages find a toolbox where nodes can be dragged into the graph. ④ Each input and output provide an "Algorithm finder" feature that lists all compatible sources or targets based on their data type.

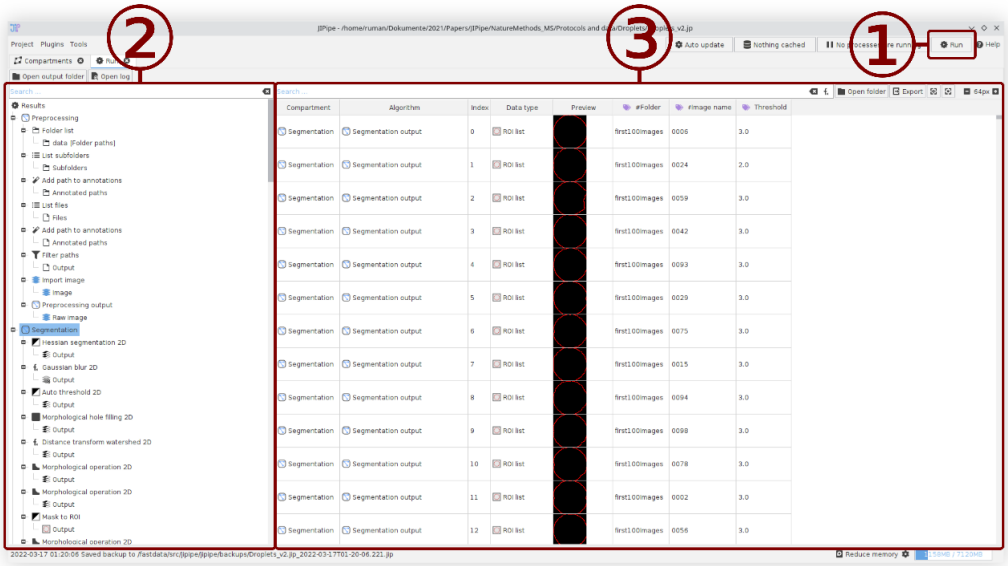
Nodes can be intuitively connected into a pipeline by creating edges between them via using the mouse. Alternatively, an algorithm finder can be used to locate nodes that match the data. Unique to *JIPipe*, all nodes are self-documenting, meaning that users can infer the functionality of the nodes and their slots without referencing a manual (see **Supplementary Figure 3.2**). The nodes are fully customizable by the user, thus simplifying the execution of multi-parameter sets.

A comprehensive context-sensitive documentation of all nodes and their parameters can be accessed any time. Alternatively, *JIPipe* includes complete documentations for nodes and data types that can be exported to HTML, PDF, or text files (see **Supplementary Figure 3.5**). To provide users of more complex nodes with a starting point, we implemented minimal examples that also reveal syntax details. Additionally, *JIPipe* nodes are capable of automated parameter validation to warn users about possibly invalid inputs.



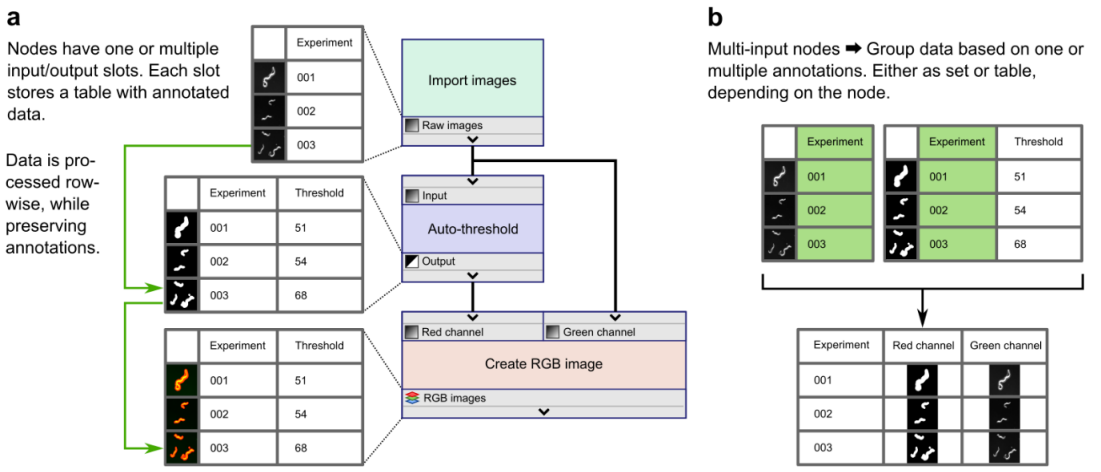
Supplementary Figure 3.5. | Integrated documentation. ① The node parameter editor provides access to a brief description of the selected node. Parameter documentations are context sensitive and displayed after hovering the parameter control. ② The "Help" menu allows access to full documentations for nodes and data types.

A pipeline is executed by clicking the "Run" button, which will automatically validate the project, and offer options for optimization and multi-threading. The user only has to set an output directory and confirm the settings. *JIPipe* will automatically execute the project and store all results in a standardized format, together with the parameters, as a full project file (see **Supplementary Figure 1.1**. Afterwards, the results are displayed in a separate interface that allows to review and export the data (see **Supplementary Figure 3.6**). Due to the standardized output format, *JIPipe* can open existing results in this viewer, even if the analysis was not applied on the same machine. To work on data interactively, *JIPipe* allows the execution of a selected node where results are cached inside the *random-access memory* (RAM) to be accessed later from within the GUI and displayed in the appropriate tool, e.g., images are displayed by *ImageJ* or other tools, or re-used by another *JIPipe* run. To improve the usability of caching, *JIPipe* comes with cache-aware viewers for common *ImageJ* data that automatically update themselves to the newest cached versions, display metadata, and allow to browse through all results efficiently.



Supplementary Figure 3.6 | Result analysis GUI. ① The interface is opened after a successful pipeline run. Alternatively, *JIPipe* can open existing directories. ② Results are organized by their compartment, node, and output slot. On selecting an entry, the corresponding data is displayed. ③ Data is displayed as table, containing information about the compartment, node, slot, index within the data table. Text and data annotations are displayed as well. The main data item is previewed.

JIPipe organizes data into tables that are associated to each slot (see **Supplementary Figure 3.7a**). Each table has one column containing binary data of a type defined by the slot, and an arbitrary number of metadata columns containing strings or other data. There are various nodes available that generate or modify the set of metadata. For example, it can be used to track biological conditions, dataset identifiers, or image properties. The flexibility of this approach allows the easy management of research data and assists users in finding and reproducing data and analysis details according to the FAIR principles[3]. Generally, nodes iterate over the rows of the table and generate one result per row; this strategy also provides the opportunity to parallelize computationally expensive workloads. Another benefit of this design is zero-cost up- and downscaling: Users only need to modify the set of input files or folders to change the scale of the analysis without the need for updating the pipeline structure. During the processing, metadata is conserved. These annotations are helpful for the postprocessing and review steps but are also actively used by various algorithms that iterate through multiple inputs or merge data (see **Supplementary Figure 3.7b**).



Supplementary Figure 3.7 | *JIPipe* data model. (a) Each node in a pipeline (right) has multiple inputs and output slots (gray box). Each slot contains a table of binary data (left), annotated with additional string columns (e.g., "Experiment", "Threshold"). A connection between two slots (black lines) leads to the data being passed to the input and processed row-wise (green arrows). The annotations are preserved. (b) A node with multiple inputs (Figure 4a, "Create RGB image") groups data (left) by testing for the equivalence of annotation sets (green highlight). The resulting grouped table (right) is processed row-wise.

4 Online training and documentation resources

We already provide a substantial amount of online documentation that simplify the process of adapting *JIPipe* into a bioimage analysis workflow, develop plugins and extensions, and to connect our software to third-party tools.

4.1 User guide and tutorials

To provide resources for new users of *JIPipe*, we created both step-by-step tutorials in text and video form, as well as documentations that guide users through the features of the user interface. As users of *ImageJ* might be new to the concept of visual programming and are possibly unaware of the benefits gained by utilizing our software, we created a video abstract that explains these aspects within three minutes (see https://www.youtube.com/watch?v=ZyI52bluWYI). All tutorials are listed on https://www.jipipe.org/tutorials/ and already include the following items:

- A step-by-step tutorial guiding through a basic image analysis task with batch processing
  - Text (25 steps): https://www.jipipe.org/tutorials/analysis/
  - Video (9:25 minutes): https://www.jipipe.org/tutorials/analysis_video/)
- A comprehensive tutorial that compares the analysis workflow between *ImageJ* and *JIPipe*
  - Video (22:36 minutes): https://www.jipipe.org/tutorials/jipipe-for-imagej-users/
- A short overview of the *JIPipe* user interface
  - Video (4:35 minutes): https://www.jipipe.org/tutorials/guide-user-interface/
- A brief explanation of *JIPipe*'s data caching feature
  - Video (4:16 minutes): https://www.jipipe.org/tutorials/guide-data-caches/
- An explanation of the graph editor features
  - Video (3:48 minutes): https://www.jipipe.org/tutorials/guide-graph-editor/
- A tutorial that explains the setup of a batch analysis and the backgrounds of *JIPipe*'s data management
  - Video (7:47 minutes): https://www.jipipe.org/tutorials/guide-batch-processing/
- A guide through the design of a custom node via the *JIPipe* GUI:
  - Text (10 steps): https://www.jipipe.org/tutorials/extension/

Information that is not covered by our tutorials is made available in the text documentation (see https://www.jipipe.org/documentation/) that covers the following topics:

- The basic concepts behind *JIPipe*
  - The basic concepts of visual programming with focus on users familiar to *ImageJ*: https://www.jipipe.org/documentation/basic-concepts/visual-programming/
  - An overview of the batch processing functionality with illustrations to explain the concepts behind it: https://www.jipipe.org/documentation/basic-concepts/batch-processing/
- Information about important GUI and *JIPipe* features related to designing pipelines
  - An overview of the graph editor user interface: https://www.jipipe.org/documentation/create-pipelines/pipeline-editor/
  - A detailed explanation of *JIPipe*'s expression system with illustrations, examples, and a list of operators and their precedence: https://www.jipipe.org/documentation/create-pipelines/expressions/
  - An explanation of the purpose of graph compartments: https://www.jipipe.org/documentation/create-pipelines/compartments/
  - A guide through the node grouping functionality: https://www.jipipe.org/documentation/create-pipelines/groups/
  - An explanation on the usage of loop nodes: https://www.jipipe.org/documentation/create-pipelines/loops/
- Guides relating to running pipelines and reviewing results
  - A brief guide on how to run a pipeline: https://www.jipipe.org/documentation/run-pipelines/run/
  - A guide through the result viewing component: https://www.jipipe.org/documentation/run-pipelines/result-analysis/
  - An overview of *JIPipe*'s data storage format: https://www.jipipe.org/documentation/run-pipelines/connect-external-software/
  - Information on how users can cache data: https://www.jipipe.org/documentation/run-pipelines/quick-run/ and https://www.jipipe.org/documentation/run-pipelines/cache/
- Information about the *ImageJ* integration and how to run *JIPipe* nodes inside *ImageJ*: https://www.jipipe.org/documentation/imagej-integration/
- An overview of *JIPipe*'s plugin list GUI: https://www.jipipe.org/documentation/plugins/
- An overview of all functionalities included in the standard *JIPipe* distribution
  - The *ImageJ* integration library: https://www.jipipe.org/documentation/standard-library/imagej-integration/
  - A guide on how to utilize macro nodes: https://www.jipipe.org/documentation/standard-library/macro-node/
  - Important remarks regarding the file system nodes: https://www.jipipe.org/documentation/standard-library/filesystem/
  - A guide through the multi-parameter feature supported by many nodes: https://www.jipipe.org/documentation/standard-library/multi-parameter/
  - Remarks about the usage of data annotations: https://www.jipipe.org/documentation/standard-library/annotations/
  - A guide through the plotting features included in *JIPipe*: https://www.jipipe.org/documentation/standard-library/plots-tables/

    - An overview of the integrated Jython and Python wrappers: https://www.jipipe.org/documentation/standard-library/jython/, https://www.jipipe.org/documentation/standard-library/python/, https://www.jipipe.org/documentation/standard-library/python/api/
    - Information about the R integration: https://www.jipipe.org/documentation/standard-library/r-integration/
    - An overview of the Cellpose nodes and information about how they are utilized: https://www.jipipe.org/documentation/standard-library/cellpose/
- Information about the creation custom *JIPipe* extensions via a graphical interface: https://www.jipipe.org/documentation/create-json-extensions/
- The usage of *JIPipe* within a command line interface: https://www.jipipe.org/documentation/cli/

4.2 Java API documentation

To aid with the continued development of *JIPipe* and to facilitate the creation of new extensions, we published documentation about *JIPipe*'s Java API. This includes the automatically generated JavaDocs that contain all classes, methods, and packages (see https://www.jipipe.org/apidocs/index.html), but also detailed guides on how to setup an extension project, create nodes, data types, parameters, and other features:

- The setup of a Java Maven project that provides features for *JIPipe*: https://www.jipipe.org/documentation-java-api/create-extension/
- An overview of the node type classes, including an example node implementation: https://www.jipipe.org/documentation-java-api/algorithm/
    - Documentation on the development of iterative multi-input nodes: https://www.jipipe.org/documentation-java-api/algorithm/iterating-algorithms/
    - An alternative multi-input node type that merges multiple data items: https://www.jipipe.org/documentation-java-api/algorithm/merging-algorithms/
    - Remarks regarding the modification of node input and outputs: https://www.jipipe.org/documentation-java-api/algorithm/slot-configuration/
    - A basic guide to defining node parameters: https://www.jipipe.org/documentation-java-api/algorithm/parameters/
    - Guidelines for creating nodes that support parallelized workloads: https://www.jipipe.org/documentation-java-api/algorithm/parallelization/
    - Definition of node types that do not have a one-to-one relationship with a Java class: https://www.jipipe.org/documentation-java-api/algorithm/custom-info/
    - A guide on creating interactive buttons in parameter lists: https://www.jipipe.org/documentation-java-api/algorithm/context-actions/
- An overview on how the Java API is used to create a new data type: https://www.jipipe.org/documentation-java-api/data-type/
    - Explanations on how to create user-selectable data importers: https://www.jipipe.org/documentation-java-api/data-type/result-ui/
    - Documentation on implementing result previews: https://www.jipipe.org/documentation-java-api/data-type/result-preview/
- A guide through the creation of a new parameter type: https://www.jipipe.org/documentation-java-api/parameter-type/
- Interfacing with *JIPipe* through its Java API to run nodes, pipelines, and projects: https://www.jipipe.org/documentation-java-api/usage-in-java/

4.3 Data and JSON API documentation

To store data and projects, *JIPipe* utilizes JSON files that follow a standardized format. This includes the format for projects (see https://www.jipipe.org/documentation-json-api/project/) and for non-Java extensions (see https://www.jipipe.org/documentation-json-api/json-extension/). A similar standardization is applied in the storage of output files to allow automated data reading and writing operations. Its highest-order implementation is the standardized format for whole-pipeline and is described in https://www.jipipe.org/documentation-data-api/pipeline-output/. The format makes use of the "data table" standard (see https://www.jipipe.org/documentation-data-api/data-table/) that makes the automated reading and writing of data and metadata possible, due to the presence of a standardized metadata file ("data-table.json", see https://www.jipipe.org/documentation-json-api/data-table/). Within the a data table directory, data items are stored in various directories that contain hierarchies of files and folders that follow a standard defined by the data type definition in Java (see https://www.jipipe.org/documentation-data-api/row-folder/). A list of available data types, associated standards and properties is given in https://www.jipipe.org/documentation-data-api/data-types/.

5 Methods

5.1 *JIPipe* dependencies

JIPipe is written in Java version 8 and utilizes libraries provided by *SciJava* (https://scijava.org/) . The full list of required libraries is shown in **Supplementary Table 1**. *JIPipe* is open source and licensed under BSD-2-Clause license.

Dependency	Version	Author
Bio-Formats	6.5.1	Linkert et al.[14]
CLIJ2	2.0.0.14	Haase et al.[1]
Feature_Detection	2.0.2	Fiji.sc
FeatureJ	2.0.0	Erik Meijering[15]
Flexmark	0.62.2	Vladimir Schneider
Guava	26.0-jre	Google Inc.
ImageJ	2.1.0	Rueden et al.[16]
ImageScience	3.0.0	Erik Meijering
ImgLib2	2.0.0-beta-46	Pietzsch et al.[17]
Jackson	2.11.0	FasterXML
Javaluator	3.0.3	Jean-Marc Astesana
JFreeChart	1.5.0	JFree.org
JFreeSVG	3.4	JFree.org
JGraphT	1.4.0	Barak Naveh
JUnit	5.7.0	JUnit Team
Jython	2.7.2	Jython Project
MorphoLibJ	1.4.1	Legland et al.[18]
MPICBG	1.3.0	Stephan Saalfeld
MSLinks	1.0.5	Dmitrii Shamrikov
MTrackJ	1.5.4	Erik Meijering
Multi-Template-Matching	-	Thomas, Gehrig[19]
OMERO	5.5.8	Allan et al.[2]
RandomJ	2.0.0	Erik Meijering
Reflections	0.9.12	ronmamo
Scijava	29.2.1	Rueden et al.[20]
SLF4J	1.7.9	QOS.ch
SwingX	1.6.1	SwingLabs

Trove4J	3.0.3	Rob Eden
Apache Commons Exec	1.3	The Apache Software Foundation
Apache Commons Compress	1.9	The Apache Software Foundation
JNA	4.5.2	Timothy Wall, Matthias Bläsing

Supplementary Table 5.1 | List of libraries used by *JIPipe*.

5.2 *JIPipe* system requirements

JIPipe was designed to run on any operating system supported by ImageJ and tested on Linux (Ubuntu 22.04), Windows (Windows 10), and macOS (macOS Sierra 10.12.6). We must note that due to our limited access to test MacOS, we can't ensure that all features will work as expected on Apple computers. We are open to contributions from the community. We recommend running our software on hardware with at least 8 GB of system memory and on a 64-bit operating system. To execute all example pipelines provided with this manuscript, at least 16 GB of memory are required. To utilize GPU processing functionality provided by CLIJ2, a graphics card supporting OpenCL 1.2 or higher and capacity to store the analyzed images must be available.

6 References

1. Haase, R. *et al.* CLIJ: GPU-accelerated image processing for everyone. *Nat. Methods* **17**, 5–6 (2020).
2. Allan, C. *et al.* OMERO: flexible, model-driven data management for experimental biology. *Nat. Methods* **9**, 245–253 (2012).
3. Svensson, C. M. *et al.* Coding of Experimental Conditions in Microfluidic Droplet Assays Using Colored Beads and Machine Learning Supported Image Analysis. *Small* **15**, 1802384 (2019).
4. Mahler, L. *et al.* Enhanced and homogeneous oxygen availability during incubation of microfluidic droplets. *RSC Adv.* **5**, 101871–101878 (2015).
5. Zang, E. *et al.* Real-time image processing for label-free enrichment of Actinobacteria cultivated in picolitre droplets. *Lab. Chip* **13**, 3707–3713 (2013).
6. Cseresnyes, Z., Kraibooj, K. & Figge, M. T. Hessian-based quantitative image analysis of host-pathogen confrontation assays. *Cytometry A* **93**, 346–356 (2018).
7. Muljajew, I. *et al.* Stealth Effect of Short Polyoxazolines in Graft Copolymers: Minor Changes of Backbone End Group Determine Liver Cell-Type Specificity. *ACS Nano* **15**, 12298–12313 (2021).
8. Hassan, M. I. A. *et al.* The geographical region of origin determines the phagocytic vulnerability of Lichtheimia strains. *Environ. Microbiol.* **21**, 4563–4581 (2019).

9. Cseresnyes, Z., Hassan, M. I. A., Dahse, H.-M., Voigt, K. & Figge, M. T. Quantitative Impact of Cell Membrane Fluorescence Labeling on Phagocytosis Measurements in Confrontation Assays. *Front. Microbiol.* **11**, 1193 (2020).
10. Stringer, C., Wang, T., Michaelos, M. & Pachitariu, M. Cellpose: a generalist algorithm for cellular segmentation. *Nat. Methods* **18**, 100–106 (2021).
11. Büttner, H. *et al.* Bacterial endosymbionts protect beneficial soil fungus from nematode attack. *Proc. Natl. Acad. Sci. U. S. A.* **118**, 2110669118 (2021).
12. Klingberg, A. *et al.* Fully Automated Evaluation of Total Glomerular Number and Capillary Tuft Size in Nephritic Kidneys Using Lightsheet Microscopy. *J. Am. Soc. Nephrol. JASN* **28**, 452–459 (2017).
13. Gerst, R., Medyukhina, A. & Figge, M. T. MISA++: A standardized interface for automated bioimage analysis. *SoftwareX* **11**, (2020).
14. Linkert, M. *et al.* Metadata matters: access to image data in the real world. *J. Cell Biol.* **189**, 777–782 (2010).
15. Meijering, E. FeatureJ. https://imagescience.org/meijering/software/featurej/.
16. Rueden, C. T. *et al.* ImageJ2: ImageJ for the next generation of scientific image data. *BMC Bioinformatics* **18**, 529 (2017).
17. Pietzsch, T., Preibisch, S., Tomančák, P. & Saalfeld, S. ImgLib2—generic image processing in Java. *Bioinformatics* **28**, 3009–3011 (2012).
18. Legland, D., Arganda-Carreras, I. & Andrey, P. MorphoLibJ: Integrated library and plugins for mathematical morphology with ImageJ. *Bioinformatics* **32**, 3532–3534 (2016).
19. Thomas, L. S. V. & Gehrig, J. Multi-template matching: a versatile tool for object-localization in microscopy images. *BMC Bioinformatics* **21**, 44 (2020).
20. Rueden, C., Schindelin, J., Hiner, M. & Eliceiri, K. SciJava Common [Software]. (2016).

4.3 Bacterial endosymbionts protect beneficial soil fungus from nematode attack

Manuskript Nr. 3

Titel des Manuskriptes: Bacterial endosymbionts protect beneficial soil fungus from nematode attack

Autoren: Hannah Büttner, Sarah P. Niehls, Koen Vandelannoote, Zoltán Cseresnyés, Oliver Sommerfeld, Benjamin Dose, Ingrid Richter, Ruman Gerst, Marc Thilo Figge, Timothy P. Stinear, Sacha J. Pidot, Christian Hertweck

Bibliographische Informationen: Büttner, H., Niehs, S. P., Vandelannoote, K., Cseresnyés, Z., Dose, B., Richter, I., ... & Hertweck, C. (2021). Bacterial endosymbionts protect beneficial soil fungus from nematode attack. Proceedings of the National Academy of Sciences, 118(37)

Der Kandidat / Die Kandidatin ist (bitte ankreuzen)

☐ Erstautor/-in, ☐ Ko-Erstautor/-in, ☐ Korresp. Autor/-in, ☒ Koautor/-in.

Status: Veröffentlicht in Proceedings of the National Academy of Sciences

Anteile (in %) der Autoren / der Autorinnen an den vorgegebenen Kategorien der Publikation

Autor/-in	Konzeptionell	Datenanalyse	Experimentell	Verfassen des Manuskriptes	Bereitstellung von Material
Hannah Büttner					
Sarah P. Niehls					
Koen Vandelannoote					
Zoltán Cseresnyés	50 %	60 %	60 %	50 %	0 %
Oliver Sommerfeld					
Benjamin Dose					
Ingrid Richter					
Ruman Gerst	20 %	30 %	30 %	20 %	0 %
Marc Thilo Figge	30 %	10 %	10 %	30 %	100 %
Timothy P. Stinear					
Sacha J. Pidot					
Christian Hertweck					
Weitere					
Summe:	100 %	100 %	100 %	100 %	100 %

In der obigen Tabelle schätzen wir lediglich den bioinformatorischen Teil (= 100%) dieser interdisziplinären Arbeit.

_____ _____

Unterschrift Kandidat/-in Unterschrift Betreuer/-in (Mitglied der Fakultät)

Bacterial endosymbionts protect beneficial soil fungus from nematode attack

Hannah Büttner[a,1], Sarah P. Niehs[a,1], Koen Vandelannoote[b], Zoltán Cseresnyés[c], Benjamin Dose[a], Ingrid Richter[a], Ruman Gerst[c,d], Marc Thilo Figge[c,d], Timothy P. Stinear[b], Sacha J. Pidot[b,2], and Christian Hertweck[a,d,2]

[a]Department of Biomolecular Chemistry, Leibniz Institute for Natural Product Research and Infection Biology–Hans Knöll Institute (HKI), 07745 Jena, Germany; [b]Department of Microbiology and Immunology, Doherty Institute, Melbourne, 3000, Australia; [c]Applied Systems Biology Research Group, Leibniz Institute for Natural Product Research and Infection Biology–Hans Knöll Institute (HKI), 07745 Jena, Germany; and [d]Institute of Microbiology, Faculty of Biological Sciences, Friedrich Schiller University Jena, 07743 Jena, Germany

Edited by Nancy A. Moran, The University of Texas at Austin, Austin, TX, and approved August 5, 2021 (received for review June 9, 2021)

Fungi of the genus *Mortierella* occur ubiquitously in soils where they play pivotal roles in carbon cycling, xenobiont degradation, and promoting plant growth. These important fungi are, however, threatened by micropredators such as fungivorous nematodes, and yet little is known about their protective tactics. We report that *Mortierella verticillata* NRRL 6337 harbors a bacterial endosymbiont that efficiently shields its host from nematode attacks with anthelmintic metabolites. Microscopic investigation and 16S ribosomal DNA analysis revealed that a previously overlooked bacterial symbiont belonging to the genus *Mycoavidus* dwells in *M. verticillata* hyphae. Metabolic profiling of the wild-type fungus and a symbiont-free strain obtained by antibiotic treatment as well as genome analyses revealed that highly cytotoxic macrolactones (CJ-12,950 and CJ-13,357, *syn.* necroxime C and D), initially thought to be metabolites of the soil-inhabiting fungus, are actually biosynthesized by the endosymbiont. According to comparative genomics, the symbiont belongs to a new species (*Candidatus* Mycoavidus necroximicus) with 12% of its 2.2 Mb genome dedicated to natural product biosynthesis, including the modular polyketide-nonribosomal peptide synthetase for necroxime assembly. Using *Caenorhabditis elegans* and the fungivorous nematode *Aphelenchus avenae* as test strains, we show that necroximes exert highly potent anthelmintic activities. Effective host protection was demonstrated in cocultures of nematodes with symbiotic and chemically complemented aposymbiotic fungal strains. Image analysis and mathematical quantification of nematode movement enabled evaluation of the potency. Our work describes a relevant role for endofungal bacteria in protecting fungi against mycophagous nematodes.

natural products | symbiosis | microbial interactions

A healthy soil nourishes plants and animals, purifies water and air, and promotes sustainable agriculture. Characteristic for highly complex and competitive soil ecosystems are the frequent and direct interactions between all soil-dwelling microorganisms, animals, and plants (1, 2), all of which need to be provided with minerals and carbon sources. Thus, carbon cycling, mainly promoted by fungal saprophytes and decomposers that release nutrients from decaying matter, plays a pivotal role for soil health (3, 4). Fungi belonging to the genus *Mortierella* are the most common soil-dwelling fungi, ubiquitously distributed in all parts of the world, inhabiting highly diverse niches including the rhizosphere and plant tissues (5–9). Owing to their ability to degrade biopolymers as well as xenobiotics, they not only deliver energy-rich carbon sources but also clear the environment from pollutants (10, 11). Typically associated with healthy soils, *Mortierella* species are recognized as valuable plant growth–promoting fungi in agriculture (9, 12).

Even so, all fungi, including *Mortierella* species, are threatened by micropredators such as nematodes (13–15). In order to oppose these predators, fungi have developed a diverse set of defense strategies. These include the production of toxic proteins and nematocidal natural products, hyphal piercing, trapping, egg parasitism, and endoparasitism (13, 16). Information on defense strategies employed by *Mortierella* species against nematodes is, however, scarce. It is known that *Mortierella globalpina* traps nematodes by means of its hyphae and penetrates the nematode's cuticula. In this way, *M. globalpina* may protect its host plants from plant-parasitic nematodes (e.g., *Meloidogyne chitwoodi*) (17). Antinematode activities have been implicated for some *Mortierella* species (18, 19), including *Mortierella alpina* [against *Meloidogyne javanica* or *Heterodera* sp. (20, 21)], but it is not a general trait of *Mortierella* (21, 22). Apart from the hyphal trapping strategy, insight into the molecular basis of the antinematode activities of *Mortierella* is missing. Furthermore, on a more general note, it is remarkable that thus far no *Mortierella* secondary metabolites have been associated with potential protective roles against nematodes.

Here, we report a so far unknown strategy of a *Mortierella* species to protect itself from nematode attack. We provide evidence that cytotoxic benzolactones initially isolated from fungal cultures are in fact produced by bacterial endosymbionts that have been overlooked thus far. We also show that the bacteria dwelling in the fungal hyphae protect their host from predatory nematodes.

Significance

Soil is a complex and competitive environment, forcing its inhabitants to develop strategies against competitors, predators, and pathogens. Identifying and understanding the molecular mechanisms has translational value for medicine, ecology, and agriculture. In this study, we show that a member of important soil-dwelling fungi (*Mortierella*) forms a tight alliance with toxin-producing bacteria (*Mycoavidus*) that live within the fungal hyphae and protect their host from nematode attack. This discovery is relevant since *Mortierella* species correlate with healthy soils and are used as plant growth–promoting fungi in agriculture. Unraveling an ecological role for fungal endosymbionts in *Mortierella*, our results contribute to the understanding of a mainspring in fungal–endobacterial symbioses and open the possibility for the development of new biocontrol agents.

Author contributions: H.B., S.P.N., and C.H. designed research; H.B., S.P.N., K.V., Z.C., B.D., I.R., R.G., and S.J.P. performed research; H.B., S.P.N., K.V., Z.C., B.D., I.R., R.G., M.T.F., T.P.S., and S.J.P. analyzed data; and H.B., S.P.N., S.J.P., and C.H. wrote the paper.

The authors declare no competing interest.

This article is a PNAS Direct Submission.

[1]H.B. and S.P.N. contributed equally to this work.

[2]To whom correspondence may be addressed. Email: christian.hertweck@leibniz-hki.de or sacha.pidot@unimelb.edu.au.

This article contains supporting information online at https://www.pnas.org/lookup/suppl/doi:10.1073/pnas.2110669118/-/DCSupplemental.

Published September 9, 2021.

MICROBIOLOGY

Results and Discussion

***Mortierella* Fungus Harbors Bacterial Endosymbionts Producing Toxic Macrolactones.** We reasoned that benzolactones CJ-12,950 and CJ-13,357 (Fig. 1*A*) (23) from cultures of *Mortierella verticillata* [synonym *Podila verticillata* (24)] could play a role as nematode defense metabolites. Although the initial report on CJ-12,950 and CJ-13,357 only stated that these compounds enhance the expression of the low-density lipoprotein receptor in human hepatocytes (23), they share the benzolactone enamide architecture with structurally related vATPase inhibitors (25, 26). Moreover, the architectures of CJ-12,950 and CJ-13,357 specifically resemble those of *Burkholderia* sp. strain B8 produced necroximes A to D (**1** to **4**), which proved to be cytotoxic (27). Since only the two-dimensional structures of CJ-12,950 and CJ-13,357 had been reported (23), we assigned their absolute configurations by examining the structural relationships with necroximes C and D. Optical rotation comparison, high-performance liquid chromatography (HPLC)–based coelution experiments and comparison of tandem mass spectrometry (MS/MS) fragmentation indicated that necroxime D (**4**) is identical to CJ-12,950, and necroxime C (**3**) is identical to CJ-13,357 (Fig. 1*B* and *SI Appendix*, Table S4). These assignments were corroborated by comparison of the NMR spectra of purified metabolites (*SI Appendix*, Table S9).

Given the bacterial origin of the necroximes (27) and related benzolactones (25, 28–30), we questioned the biosynthetic capability of *M. verticillata* and sought to identify the true producer. Since several *Mortierella* spp. have been reported to live in symbiosis with bacteria (31, 32), we suspected an endosymbiont to be the true source of **3** and **4**. Yet, a 2018 report investigating the prevalence of Burkholderiaceae-related bacteria within *Mortierella* spp. stated that strain NRRL 6337 was devoid of endosymbionts (32). Nonetheless, we re-examined the same strain for endosymbionts by staining fungal hyphae with the chitin-binding Calcofluor White dye, and tentative endobacteria with the nucleic acid dye Syto9 Green (Fig. 1*C*). Fluorescence microscopy revealed the presence of endosymbiotic organisms in *M. verticillata* NRRL 6337 (*SI Appendix*, Fig. S1).

To identify the observed bacterial endosymbionts, we cut a small piece of fungal mycelium and extracted holobiont DNA, followed by PCR amplification of the 16S ribosomal DNA (rDNA) region using universal primers. Sequencing of the 16S rDNA region (*SI Appendix*, Table S1) and BLAST analysis indicated that the symbiont of *M. verticillata* NRRL 6337 is a *Mycoavidus* species. Notably, members of this genus have been reported as symbionts of soil-dwelling fungi (32). So far, the full genomes of only three *Mycoavidus cysteinexigens* strains from *Mortierella elongata* and *Mortierella parvispora* have been sequenced (31, 33–35). PCR-amplified bacterial 16S rDNA sequences from other *Mortierella* fungi, however, revealed further *Mycoavidus* endosymbionts with three phylogenetically distant clades (*Mortierella*-associated *Burkholderia*-related endosymbiont [MorBRE] groups A to C) (32). Through phylogenetic analysis, we found that the *Mycoavidus* symbiont of *M. verticillata* NRRL 6337 falls into MorBRE group A (Fig. 1*D* and *SI Appendix*, Fig. S3) comprising symbionts of *Mortierella humilis*, *Mortierella gamsii*, *Mortierella basiparvispora*, and *M. elongata* (*M. cysteinexigens*). To better understand the occurrence of *Mycoavidus* endosymbionts in *M. verticillata* strains, we investigated five additional *M. verticillata* strains for the presence of endosymbionts. Amplification of the 16S rDNA regions from gDNA of symbionts of these strains revealed a conserved occurrence of *Mycoavidus* endosymbionts in *M. verticillata* strains. Interestingly, these additional endosymbionts all fall into another phylogenetic group together with *Burkholderia* sp. strain B8. Furthermore, analysis of the metabolic profiles of the respective fungi did not show any production of necroximes (Fig. 1 *D* and *E*). This finding shows that endosymbionts may frequently occur in *Mortierella* and other species of the order Mucorales, but they can be phylogenetically different.

To clarify whether bacterial endosymbionts are the true producers of **3** and **4**, we aimed at curing *M. verticillata* NRRL 6337 of its symbiont through the addition of antibiotics (36). Over the course of several months, we subcultivated the fungal strain on agar plates containing kanamycin, ciprofloxacin, or chloramphenicol. During treatment, changes of the fungal growth were noticeable (Fig. 1*F*). Finally, we confirmed the absence of the symbionts by fluorescence staining, microscopic inspection, and PCR analysis (*SI Appendix*, Figs. S2 and S4). The metabolic profiling of the symbiont-free fungal strain by liquid chromatography (LC) combined with high-resolution electrospray ionization revealed the complete absence of **3** and **4** (Fig. 1*B*). These findings indicate that *Candidatus* Mycoavidus necroximicus is the true producer of the benzolactones.

***Ca.* M. necroximicus Dedicates 12% of Its Genome to Secondary Metabolism.** To gain insight into the symbiont's biosynthetic potential, with particular focus on the molecular basis of necroxime biosynthesis, we aimed at sequencing the genome of the endosymbiont. Attempts to isolate and cultivate the endosymbiont in the absence of the fungal host, however, proved to be futile. Methods previously used to axenically cultivate similar fungal endobacteria did not enable growth of the endosymbionts (33, 37), indicating a strong dependence of the bacterial symbiont on the host environment. Thus, we sought to enrich the symbiotic bacteria for DNA isolation. Initially, physical disruption of the host's mycelium resulted in high levels of contamination with fungal DNA, which complicated the assembly of the endosymbiont's genome. Eventually, we succeeded in retrieving a bacterial cell pellet by filtration and centrifugation of the turbid supernatant of shaking cultures in baffled flasks and isolated the genomic DNA from resuspended bacteria.

The genome of the bacterial endosymbiont was sequenced using a combination of Oxford Nanopore MinION and Illumina NextSeq sequencing, and both data sets were used to generate a hybrid genome assembly. Of the 118 contigs, a single 2.4 Mb contig of putative bacterial origin was identified through homology searches using the *Mycoavidus*-like 16S rDNA sequence previously amplified from *M. verticillata* NRRL 6337. Following trimming of overlapping ends (suggesting a circular chromosome) the final 2.2 Mb contig was found to contain 1,768 CDS, 6 rRNAs, 42 tRNAs, and a GC content of 50.6% (genome accession number: PRJNA733818). The 16S rDNA sequence of the new strain has 98.82% nucleotide identity to *M. cysteinexigens* B1-EB^T (33). Even so, genomic comparisons showed an average nucleotide identity of only 81.85% across the two genomes. By current standards for molecular species discrimination, the newly identified *Mortierella* endosymbiont should be considered a new species (*Ca.* M. necroximicus) (38, 39).

By comparative genomic analyses, we noted that the genomes of the two endofungal strains AG77 and B1-EB^T isolated from *M. elongata* (33, 35) are 400 to 500 kb larger than the genome of *Ca.* M. necroximicus. Only the genome of strain B2-EB isolated from *M. parvispora* (34) is smaller (~500 kb) than the genome of *Ca.* M. necroximicus (2.2 Mb). When investigating shared protein orthologs, we noted that a core genome encoding 1,164 proteins exists among the four genomes at the 70% identity level (Fig. 2*A*). However, a further all-versus-all comparison showed B1-EB^T and AG77 to be the most closely related as they share ~75% of their deduced proteome. The B2-EB and *Ca.* M. necroximicus strains are more distantly related to B1-EB^T and AG77, as well as each other, with only a small number of proteins shared exclusively with either B1-EB^T (20 and 17 proteins, respectively) or AG77 (17 and 22 proteins, respectively) (Fig. 2*B*).

Whereas biosynthetic gene clusters (BGCs) are present in the genomes of all four studied *Mortierella* symbionts, antiSMASH

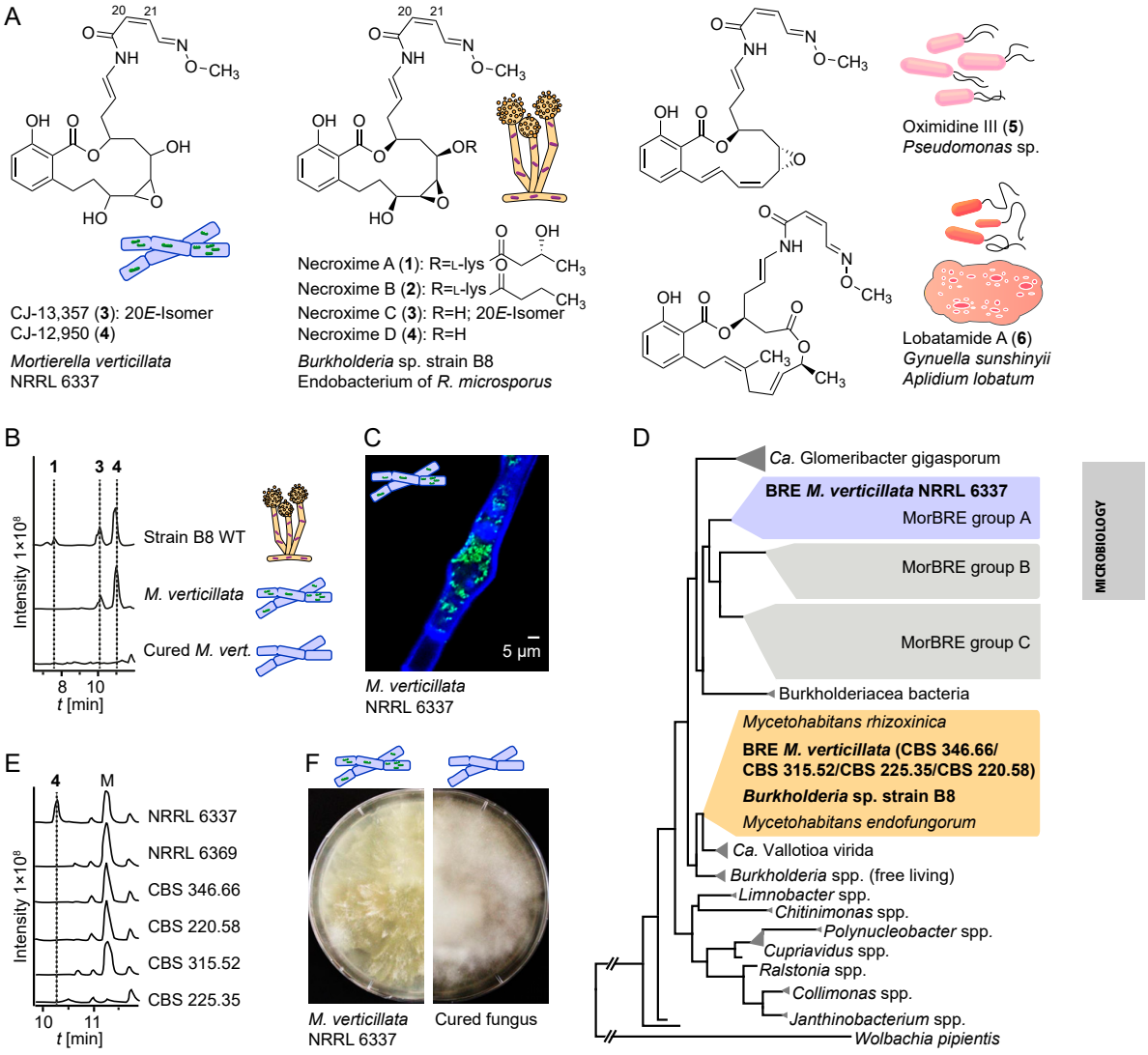


Fig. 1. Bacterial origin of cytotoxic benzolactones from *M. verticillata* cultures. (*A*) Cytotoxic lactone compounds assigned to endofungal symbionts from the fungus *R. microsporus* (**1**–**4**), *M. verticillata* (**3**–**4**), *Pseudomonas* sp. (**5**), and a tunicate and the bacterium *Gynuella sunshinyii* (**6**). (*B*) Metabolic profiles of extracts from *Burkholderia* sp. strain B8 and *M. verticillata* NRRL 6337 as symbiont or cured strain as total ion chromatograms in the negative mode. (*C*) Fluorescence micrograph depicting endosymbionts living in the fungal hyphae; staining with Calcofluor White and Syto9 Green. (*D*) Phylogenetic relationships of *Mortierella* symbionts, *Burkholderia* sp. strain B8, and other bacteria based on 16S rDNA. BRE, *Burkholderia*-related endosymbiont of *Mortierella* spp. (*E*) Metabolic profiles of extracts from *M. verticillata* NRRL 6337 and other necroxime-negative *M. verticillata* strains analyzed for endosymbionts in this study as total ion chromatograms in the negative mode. M, medium component. (*F*) Growth of symbiotic *M. verticillata* NRRL 6337 in comparison to the cured strain.

analysis (40) revealed that the biosynthetic potential for secondary metabolites is by far the greatest in *Ca.* M. necroximicus (Fig. 2*C*). Despite the relatively small genome for *Mycoavidus* standards, ~12% of its protein-encoding capacity is dedicated to natural product biosynthesis. We identified nine nonribosomal peptide synthetase (NRPS) gene clusters, two polyketide synthase (PKS) gene clusters, two hybrid PKS/NRPS gene clusters, and five other BGCs (Fig. 2*D*). Notably, several large PKS and NRPS gene loci present in the *Ca.* M. necroximicus genome are absent in the genomes of strains B1-EB^T, B2-EB, and AG77 (Fig. 2*B*). This BGC list includes a cryptic BGC (Mcyst_0009–0017) encoding a PKS/NRPS hybrid that shows high similarity to the necroxime assembly line from *Burkholderia* sp. strain B8 (97% coverage, ~70% amino acid identity), which has been unequivocally

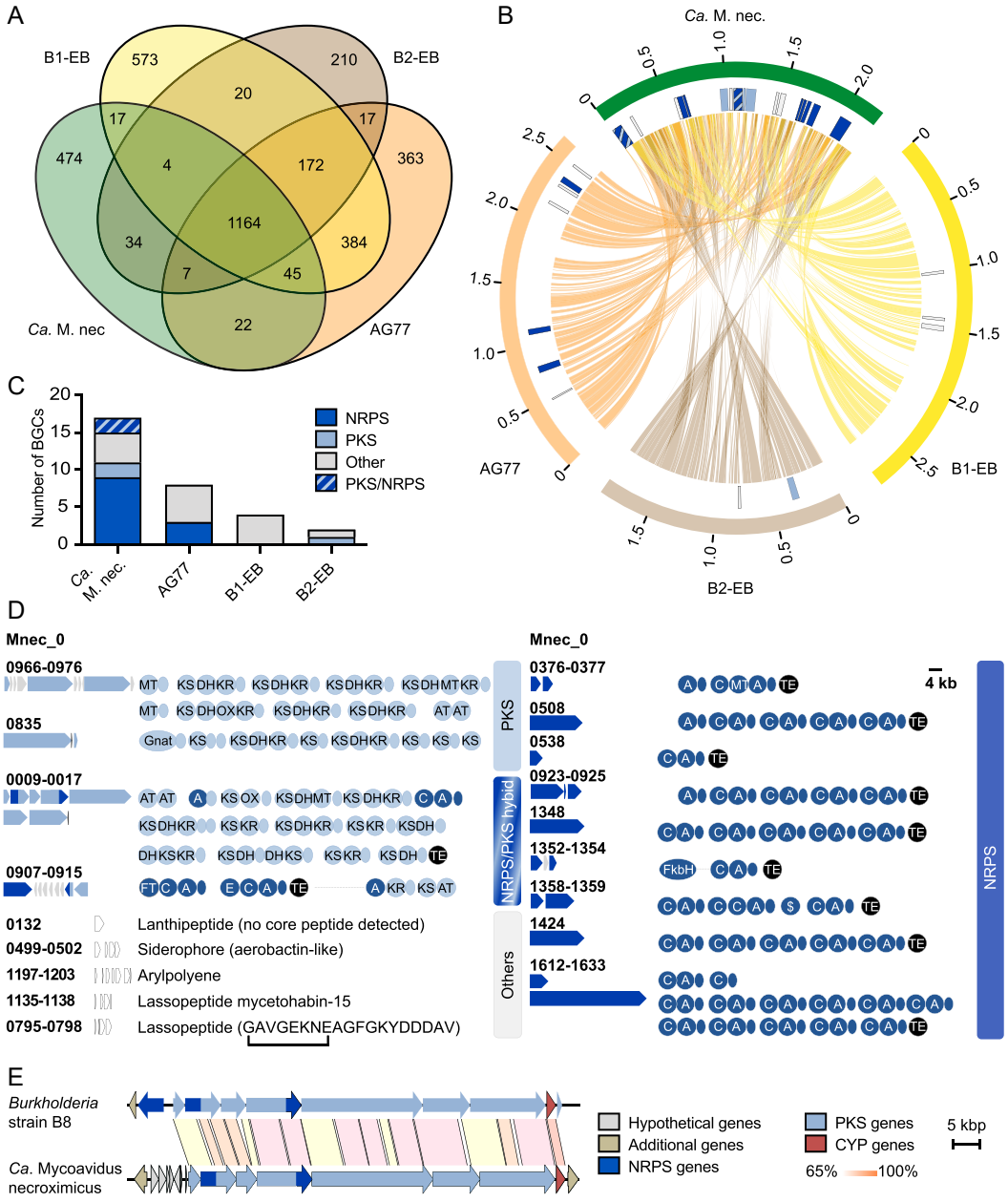


Fig. 2. Comparative genomic analyses of *Mycoavidus* spp. (*A*) Number of orthologous proteins among the four *Mycoavidus* strains at 70% identity. (*B*) Circos plot of shared protein orthologs, and secondary metabolite loci (detected by antiSMASH v5) in *Mycoavidus* genomes. Outer blocks (orange, brown, yellow, green) represent genome sizes, while the inner blocks represent genomic positions of secondary metabolite loci. Lines linking the three genomes show position of genes whose proteins are orthologous at 70% identity. Depicted are the genome sequences of *M. cysteinexigens* strains AG77, B1-EB, B2-EB, and *Ca.* M. necroximicus (*Ca.* M. nec.). (*C*) Number of gene clusters putatively coding for natural products in *Mycoavidus* spp. detected by antiSMASH and by manual assignment. (*D*) BGCs and their encoded assembly lines identified from the endofungal *Ca.* M. necroximicus are displayed. A, adenylation; AT, acyltransferase; C, condensation; DH, dehydratase; E, epimerization; Gnat, GCN5-related *N*-acetyltransferase; KR, ketoreductase; KS, ketosynthase; MT, methyltransferase; OX, oxygenase; TE, thioesterase domains. Acyl carrier (light blue) and peptidyl carrier proteins (dark blue) are shown as circles without designators. (*E*) Homologous benzolactone BGCs in the genome of *Burkholderia* strain B8 and *Ca.* M. necroximicus.

linked to necroxime biosynthesis by targeted gene knockouts (Fig. 2*E*) (27). The only major difference between the two BGCs is the NRPS gene *necA*, which is missing in the genome of *Ca.* M. necroximicus. This finding is in full agreement with the current biosynthetic model, since NecA is responsible for the attachment of the peptide side chain in **1** (Fig. 1*A*) (27), which is absent in **3** and **4**. Furthermore, the architecture of the encoded PKS/NRPS modules is perfectly in line with the biosynthesis of the benzolactone enamide backbone of **3** and **4**. Based on these *in silico* predictions, we inferred that this PKS/NRPS hybrid gene cluster codes for the biosynthesis of **3** and **4** (*SI Appendix*, Figs. S6 and S8). Together with the metabolic profiling of the cured fungal strain, these data indicate that the bacterial endosymbionts, not the fungus, are the true producers of the benzolactones **3** and **4**. CJ-12,950 and CJ-13,357 are thus important additions to the small group of natural products that were believed to be fungal metabolites but are actually produced by bacterial endosymbionts; rhizoxins (41) and rhizonins (42) from symbionts of *Rhizopus microsporus* (43), and endolides from *Stachylidium bicolor* (44). From an ecological viewpoint, it is remarkabe that endosymbiotic bacteria were identified as the true producers of the virulence factor of the rice-seedling blight fungus *R. microsporus* (36, 37, 45). Given the different ecological context of *Mortierella*, however, we assumed that the necroximes may have another function in microbial interactions.

Necroximes Protect the Fungal Host from Nematode Attacks. To learn more about the potential role of necroximes (**3** and **4**) in the ecological context of the *Mortierella–Mycoavidus* symbiosis, we investigated whether these toxins could impair the growth of, or even kill, competitors. Therefore, we considered that the common natural habitat of *Mortierella* species, including *M. verticillata* NRRL 6337, is soil, and that microbial survival in the soil environment is not only determined by the capacity to grow under harsh conditions but also by the ability to defend oneself from (micro)predators (46). Among the most abundant fungal predators are nematodes, which share the same soil habitat as *Mortierella* (47).

To determine if **3** and **4** or any other endobacteria-derived substance have anthelmintic activity, we first performed a viability assay against the model organism, *Caenorhabditis elegans* (48). We cultivated both cured (*Mycoavidus*-free) and symbiotic *M. verticillata* NRRL 6337 on potato dextrose agar (PDA agar). Cultures were extracted, and each extract was fractionated by preparative HPLC. The individual fractions (F1 to F9) were subsequently tested against *C. elegans*. Anthelmintic activity in this assay was determined by the ability of *C. elegans* to feed on a supplied *Escherichia coli* food source in the presence of the different fractions. Consumption of bacteria indicates unimpeded nematodes, whereas growth of *E. coli* indicates that the nematodes are negatively affected by the added substances (Fig. 3*A*). Notably, all fractions of the extract obtained from the cured strain culture were found to be inactive in the *C. elegans* assay. In contrast, we observed a marked nematocidal activity of fraction 6 from the extract of the symbiotic fungus. By LC/MS measurements we confirmed the presence of **3** and **4** in the active fraction. In order to determine the anthelmintic potency of the major metabolite **4**, we performed the viability assay against *C. elegans* using increasing concentrations of the pure substance and determined an inhibitory concentration at 50% (IC_{50}) value of 11.3 $\mu g \cdot mL^{-1}$ (24.66 μM) (Fig. 3*B*). Interestingly, the amount of isolated necroximes from fungal cultures grown on agar plates is ~11 $\mu g \cdot mL^{-1}$. Assuming that the actual concentration in fungal hyphae is slightly higher due to an uneven diffusion into the agar and some loss during the purification steps, we conclude that the concentrations inside and around the fungal mycelium are sufficiently high to fully protect it from mycophagous nematodes.

In order to corroborate a potential host-protective role of the symbiont-derived toxin, we next focused on a fungivorous nematode. Therefore, we selected *Aphelenchus avenae*, a predator using a stylet to feed on fungi, which pierces the fungal cell wall and allows the fungivore to ingest the fungal cytoplasm (49, 50). Sharing the same soil habitat, *A. avenae* represents a realistic predator of *Mortierella* spp. (51). To investigate the effects of symbiotic and cured *M. verticillata* strains on the feeding behavior and survival of *A. avenae*, we determined the number of animals that were harvested from fungal–nematodal cocultures. In addition, we compared the mobility ratios of the nematodes in correlation to the presence or absence of the bacterial symbiont. As a control, we employed the symbiont-bearing, but necroxime-negative, *M. verticillata* strain CSB 225.35 (Fig. 1*E*), thus ruling out an influence solely based on the presence of bacterial symbionts.

To determine nematodal propagation rates, we inoculated plate cultures of symbiotic and cured fungi with *A. avenae* and cocultivated both organisms for 17 to 24 d (three biological triplicates). Subsequently, nematodes were isolated from the cocultivation plates by Baermann funneling (52), transferred onto water-agar plates, and counted by stereomicroscopic visualization. We found that significantly fewer nematodes are able to grow in the presence of the necroxime-producing endosymbionts (Fig. 3*C* and *SI Appendix*, Fig. S9 and Tables S5 and S6).

To scrutinize the effect of the toxin on the fitness of the fungivorous nematodes, we harvested the animals from cocultures and determined their movement—and thus the mobility ratios—by image analysis and mathematical quantification (Fig. 3*D*). Using stereoscopic time series to track their movement, we compared the area covered by each moving nematode during the time series to the area covered solely by its body without movement, allowing us to differentiate active (living) from inactive (dead or paralyzed) animals. Analyzing a minimum of 176 nematodes from three independent experiments, we observed a significant decrease in the mobility ratio of nematodes grown on symbiotic *M. verticillata* NRRL 6337 compared to cured NRRL 6337 cultures and to necroxime-negative CSB 225.35 cultures (Fig. 3*E* and *SI Appendix*, Fig. S10 and Tables S7 and S8).

HPLC analyses of the plate extracts detected necroximes only in symbiotic cultures of NRRL 6337 but not in cured strains or CBS 225.35, correlating once again the toxins with reduced numbers and lower fitness of the nematodes. To unambiguously assign the nematocidal activity in the propagation assay to the necroximes, we repeated the *A. avenae* assay with the cured *Mortierella* strain and chemically complemented the major toxin. Specifically, we overlaid the cured strain NRRL 6337 with solutions of **4** in increasing concentrations (25 μM [IC_{50}], 50 μM, 109 μM, and 219 μM). We then compared *A. avenae* propagation in necroxime-complemented cultures to untreated cured as well as symbiotic fungi by microscopic examination after two weeks of coincubation. For cultures supplemented with 25 μM or 50 μM of **4**, we noted a moderate reduction of nematode propagation, whereas in cultures supplemented with 109 μM or 219 μM of **4** the presence of nematodes in the fungus was abolished (Fig. 3*F* and *SI Appendix*, Figs. S11 and S12). The elevated concentrations compared to the IC_{50} value can be explained due to an uneven distribution of **4** into deeper layers of the hydrophobic fungal colony and the ongoing growth of fungal hyphae, which were not wetted with toxin solution. Nonetheless, these experiments unambiguously verified that the chemical complementation restores the anthelmintic effect. Thus, we uncovered an important role of a natural product in the complex tripartite interplay of symbiont, host, and (micro)predator (Fig. 3*G*).

Conclusions

In this study, we uncovered a previously overlooked bacterial endosymbiont that protects the important soil-dwelling fungus *M. verticillata* from a fungivorous nematode. Comparative genomics indicate that the yet unculturable bacterial symbionts belong to a new species that is endowed with a high biosynthetic potential. Through metabolic profiling of the symbiotic wild type

MICROBIOLOGY

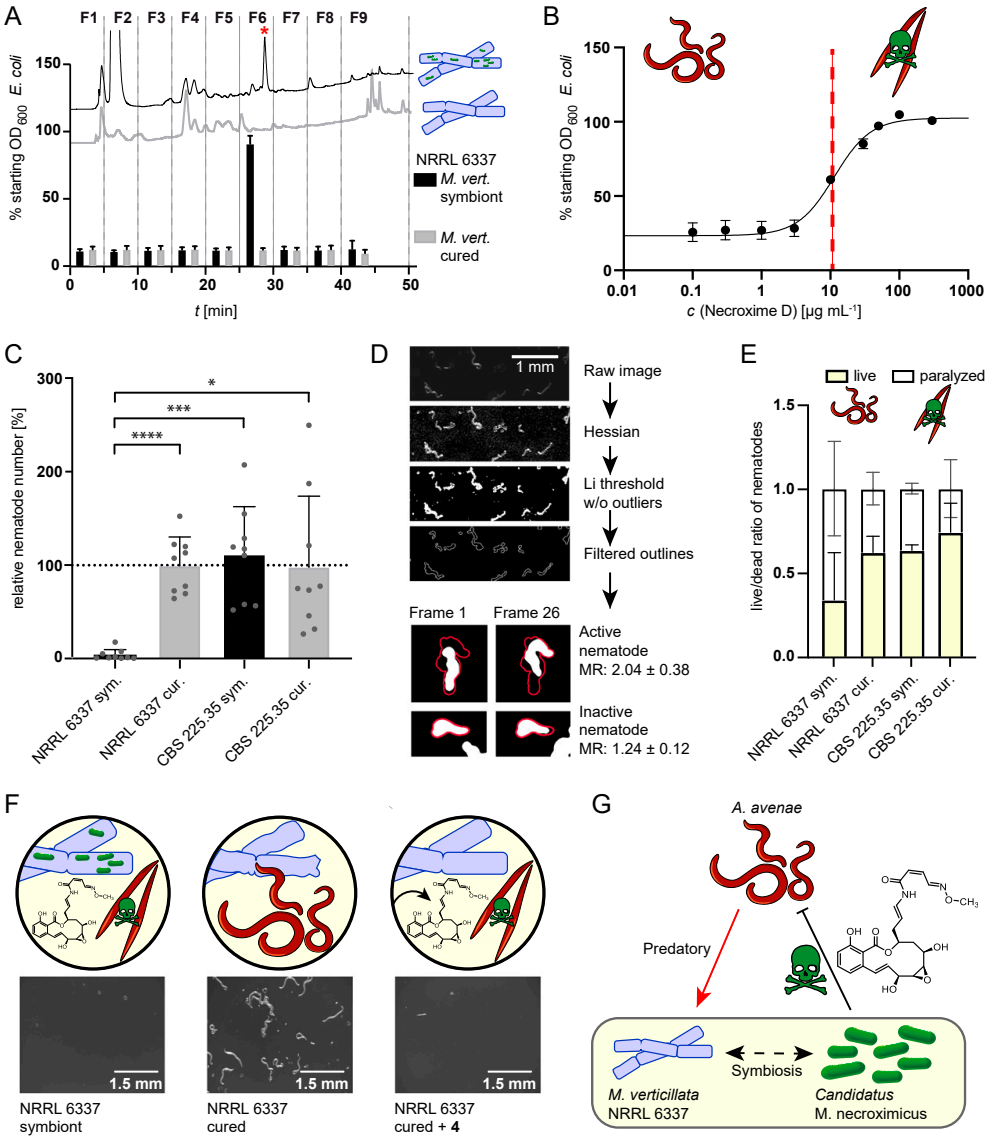


Fig. 3. Nematocidal activity of symbiont-derived toxins. (*A*) Viability assay of *C. elegans* in presence of extract fractions of symbiotic and cured *M. verticillata* NRRL 6337. HPLC profiles of extracts are shown with corresponding effect on nematodes, measured as effect on the *E. coli* optical density (OD). When nematode growth is impaired by the fraction, *E. coli* cells are not consumed, and thus the OD_{600} is not altered (error bars represent mean of three biological replicates). The red asterisk represents **4**. (*B*) Toxicity screening of **4** against *C. elegans*. The red line marks IC_{50} at 11.3 µg · mL^{-1} (24.66 µM; 95% CI, 21.45 to 28.37 µM; error bars as mean of five biological replicates). (*C*) Nematode counts from propagation assay of *M. verticillata* and *A. avenae* cocultures. Bars represent relative nematode numbers compared to the mean of the nematode count from cured *M. verticillata* NRRL 6337 cultures. cur., cured; sym., symbiotic. **P* < 0.02; ****P* < 0.001; *****P* < 0.0001. Data represent three biological replicates with three technical replicates each. (*D*) Workflow of image analysis and mathematical evaluation of *A. avenae* mobility in fungal–nematodal coincubations. Processing of time series is demonstrated by one time frame. Exemplary images of nematodes from two time frames (frame 1 and 26) are shown to illustrate differences in motility. Results of calculated mobility ratios (MR) were used for live or paralyzed/dead categorization. (*E*) Results of image analysis and mathematical quantification of nematode movement. Bars show ratio between moving/living nematodes and paralyzed/dead nematodes, which were harvested from cocultures of *A. avenae* with symbiotic *M. verticillata* NRRL 6337 cultures, cured NRRL 6337 cultures, or CSB 225.35 cultures. Numbers and error bars were calculated from minimal 176 worms from three biological replicates. (*F*) Stereomicroscopic images and schematic picture of chemical complementation assay with a magnitude of 25×. Sample of nematodes harvested from plates containing symbiotic, cured, or cured and with **4** chemically complemented *M. verticillata* NRRL 6337 cultures. (*G*) Schematic summary of tripartite interaction between fungal host, bacterial endosymbiont, and mycophagous nematodes.

and cured aposymbiotic fungi, we provide evidence that the endofungal bacteria are the true producers of highly toxic macrolides that were previously believed to be fungal metabolites. Importantly, these compounds (necroximes) efficiently protect the host from nematode attack, as demonstrated by coculture experiments, chemical complementation, and image analyses. Thus, this work not only reveals an ecological role of endofungal bacteria but also introduces a strategy to ward off micropredators. Consequently, the bacterial biosynthesis of necroximes provides an advantage of the fungal–bacterial alliance over other aposymbiotic or necroxime-negative symbiotic *M. verticillata* strains in the soil niche. Beyond inspiring the discovery of related tactics in symbioses, our findings may set the basis for new biocontrol agents, with the prospect of shielding plant hosts from plant-pathogenic nematodes.

Materials and Methods

Isolation of Natural Products. For **3** and **4** isolation, *M. verticillata* NRRL 6337 was cultivated on PDA plates (Bacto, BD) at 26 °C. The culture was extracted twice with 1:1 volume of ethyl acetate overnight. The organic phase was concentrated under reduced pressure and the residue was dissolved in methanol. The extracts were prefractionated on an open Sephadex LH-20-column with methanol as eluent. Necroxime-containing fractions were further purified with a preparative HPLC under following conditions: A, H_2O + 0.01% TFA; B, methanol; and 15 to 100% B in 35 min, 15 mL · min^{-1} [Phenomenex, Luna, 10 μm, C18(2), 100 Å, 250 × 21.2 mm]. NMR analysis was carried out on a 600 MHz Avance III Ultra Shield (Bruker), and signals were referenced to the residual solvent signal (DMSO-d_6).

Identification of Endosymbionts in *M. verticillata*. For the preparation of cured fungal strains, fungi were continuously subcultivated at 24 °C on PDA plates containing 40 μg · mL^{-1} ciprofloxacin or 50 μg · mL^{-1} kanamycin for several months. After phenotypic changes were observed by eye, an agar plate of each fungal culture was extracted with 20 mL ethyl acetate and controlled for the absence of **3** and **4** by LC/MS. Final verification of the cured fungal strains was performed by fluorescence staining (Calcofluor White Stain [Sigma] and SYTO 9 Green [Invitrogen]).

Genome Assembly for *Ca.* M. necroximicus. *M. verticillata* NRRL 6337 was grown in MM9 medium (53) and orbitally shaken at 160 rpm and 26 °C. The turbid supernatant, containing bacteria from disrupted hyphae, was twice filtered through a membrane (pore diameter, 40 μm) and centrifuged (12,000 × *g*, 25 °C, 10 min) until a stabile pellet occurred. The genomic DNA was extracted with the MasterPure DNA Purification Kit (Epicentre). For long-read sequencing on the MinION platform, DNA quality was evaluated by pulsed-field gel electrophoresis and prepared for sequencing according to the protocol of the Ligation Sequencing kit (Oxford Nanopore). DNA was loaded onto a single MinION flow cell, and data were collected over a 72-h period. DNA was prepared for sequencing on the Illumina NextSeq platform using the Nextera XT DNA preparation kit (Illumina) with ×150 bp paired end chemistry and with a targeted sequencing depth of >50×. Combined MINion and Illumina sequencing data were assembled using the Unicycler hybrid assembler (54) to form a single contig 2.2 Mb containing a 98.82% match to the *M. cysteinexigens* rDNA gene. The evaluation of secondary metabolite loci was performed with antiSMASH version 5 (55).

Nematode Assays. Liquid assays for active-fraction determination and potency assessment against *C. elegans* were conducted as previously described (48). For *A. avenae* coincubation assay, an aliquot of hyphae of each tested *Mortierella* strain was transferred to a PDA plate and incubated at 24 °C overnight. Nematodes were sterilized and starved. After one washing step with K-medium, nematodes were resuspended in 300 μL K-medium and aliquots of 50 μL were distributed onto the fresh fungal cultures. Plates were dried and controlled for living nematodes before they were incubated for 17 to 24 d at 20 °C. For the evaluation, nematodes were harvested via Baermann funneling (56). Funneled *A. avenae* were transferred on 1.5% water-agar plates containing 200 mM geneticin and 50 μg · mL^{-1} kanamycin overnight and subsequently monitored with a Zeiss Axio Zoom.V16 Stereomicroscope for worm count and bioinformatics (https://www.jipipe.org/). Remaining plates were extracted with ethyl acetate to control the metabolite production and processed as described before. For the *A. avenae* chemical complementation assay, an aliquot of hyphae of the respective fungus was transferred into 12-well plates filled with 1 mL PDA and incubated overnight. The **4** dissolved in 200 μL 50% MeOH was applied and evaporated at room temperature. Nematode suspensions of 50 μL were distributed onto the fungi, dried, and coincubated for 14 d at 20 °C. For evaluation, the coculture was removed from the well and washed in 5 mL K-medium overnight. The mixture was filtered through miracloth (Merck) to avoid agar carryover and left at 4 °C for 1 h. The remaining worms were transferred onto 6-well plates containing 5 mL 1.5% water-agar with 200 mM geneticin and 50 μg · mL^{-1} kanamycin. After the plates were dried, the worm count from each plate was assessed with a Zeiss Axio Zoom.V16 Stereomicroscope.

Data Availability. Genome sequence data have been deposited in GenBank (PRJNA733818). The 16S rDNA sequences of the *Mortierella* endosymbionts were deposited at the NCBI database (BRE_MvertCBS_346.66: MZ330684; BRE_MvertCBS_220.58: MZ330685; BRE_MvertCBS_225.35: MZ330686; BRE_MvertCBS_315.52: MZ330687; BRE_MvertCBS_100561: MZ330688).

ACKNOWLEDGMENTS. *Mortierella* strains were supplied by ARS Culture Collection (NRRL) and the Jena Microbial Resource Collection. *C. elegans* was provided by the Caenorhabditis Genetics Center, which is funded by NIH Office of Research Infrastructure Programs (P40 OD010440). *A. avenae* was received as a kind gift from Prof. Dr. M. Künzler (ETH Zürich). We thank E. Bratovanov (HKI) for helpful discussions. Assistance by K. Martin and S. Linde (HKI) is gratefully acknowledged. H.B. and S.P.N. were funded by the Deutsche Forschungsgemeinschaft (DFG; German Research Foundation) Project No. 239748522 SFB 1127, the Cluster of Excellence "Balance of the Microverse," and also the Leibniz Award (to C.H.). Z.C. and M.T.F. were funded by the DFG Project No. 316213987 SFB 1278 (Z01). I.R. acknowledges financial support from the European Union Horizon 2020 Research and Innovation Program under the Marie Sklodowska-Curie grant agreement No. 794343. R.G. was funded by the International Leibniz Research School for Microbial and Biomolecular Interactions Jena.

1. N. Fierer, Embracing the unknown: Disentangling the complexities of the soil microbiome. *Nat. Rev. Microbiol.* **15**, 579–590 (2017).
2. R. D. Bardgett, W. H. van der Putten, Belowground biodiversity and ecosystem functioning. *Nature* **515**, 505–511 (2014).
3. L. Tedersoo *et al.*, Fungal biogeography. Global diversity and geography of soil fungi. *Science* **346**, 1256688 (2014).
4. E. Ozimek *et al.*, Synthesis of indoleacetic acid, gibberellic acid and ACC-deaminase by *Mortierella* strains promote winter wheat seedlings growth under different conditions. *Int. J. Mol. Sci.* **19**, 3218 (2018).
5. D. Zhou *et al.*, Deciphering microbial diversity associated with *Fusarium* wilt-diseased and disease-free banana rhizosphere soil. *BMC Microbiol.* **19**, 161 (2019).
6. J. Yuan *et al.*, Predicting disease occurrence with high accuracy based on soil macroecological patterns of *Fusarium* wilt. *ISME J.* **14**, 2936–2950 (2020).
7. D. Liu, H. Sun, H. Ma, Deciphering microbiome related to rusty roots of *Panax ginseng* and evaluation of antagonists against pathogenic *Ilyonectria*. *Front. Microbiol.* **10**, 1350 (2019).
8. S. Edgington, E. Thompson, D. Moore, K. A. Hughes, P. Bridge, Investigating the insecticidal potential of *Geomyces* (Myxotrichaceae: Helotiales) and *Mortierella* (Mortierellacea: Mortierellales) isolated from Antarctica. *Springerplus* **3**, 289 (2014).
9. E. Ozimek, A. Hanaka, *Mortierella* species as the plant growth-promoting fungi present in the agricultural soils. *Agriculture* **11**, 7 (2021).
10. L. Ellegaard-Jensen, J. Aamand, B. B. Kragelund, A. H. Johnsen, S. Rosendahl, Strains of the soil fungus *Mortierella* show different degradation potentials for the phenylurea herbicide diuron. *Biodegradation* **24**, 765–774 (2013).
11. J. Zeng *et al.*, Lignocellulosic biomass as a carbohydrate source for lipid production by Mortierella isabellina. *Bioresour. Technol.* **128**, 385–391 (2013).
12. F. Li *et al.*, *Mortierella elongata*'s roles in organic agriculture and crop growth promotion in a mineral soil. *Land Degrad. Dev.* **29**, 1642–1651 (2018).
13. M. Künzler, How fungi defend themselves against microbial competitors and animal predators. *PLoS Pathog.* **14**, e1007184 (2018).
14. J. H. J. Leveau, G. M. Preston, Bacterial mycophagy: Definition and diagnosis of a unique bacterial-fungal interaction. *New Phytol.* **177**, 859–876 (2008).
15. S. Zhang, R. Mukherji, S. Chowdhury, L. Reimer, P. Stallforth, Lipopeptide-mediated bacterial interaction enables cooperative predator defense. *Proc. Natl. Acad. Sci. U.S.A.* **118**, e2013759118 (2021).
16. T. Degenkolb, A. Vilcinskas, Metabolites from nematophagous fungi and nematicidal natural products from fungi as an alternative for biological control. Part I: Metabolites from nematophagous ascomycetes. *Appl. Microbiol. Biotechnol.* **100**, 3799–3812 (2016).
17. M. J. DiLegge, D. K. Manter, J. M. Vivanco, A novel approach to determine generalist nematophagous microbes reveals *Mortierella globalpina* as a new biocontrol agent against *Meloidogyne* spp. nematodes. *Sci. Rep.* **9**, 7521 (2019).
18. O. Topalović, M. Hussain, H. Heuer, Plants and associated soil microbiota cooperatively suppress plant-parasitic nematodes. *Front. Microbiol.* **11**, 313 (2020).
19. W. Qiu *et al.*, Organic fertilization assembles fungal communities of wheat rhizosphere soil and suppresses the population growth of *Heterodera avenae* in the field. *Front. Plant Sci.* **11**, 1225 (2020).

20. T. A. Al-Shammari, A. H. Bahkali, A. M. Elgorban, M. T. El-Kahky, B. A. Al-Sum, The use of *Trichoderma longibrachiatum* and *Mortierella alpina* against root-knot nematode, *Meloidogyne javanica* on tomato. *J. Pure Appl. Microbiol.* **7**, 199–207 (2013).
21. S. Meyer *et al*., Activity of fungal culture filtrates against soybean cyst nematode and root-knot nematode egg hatch and juvenile motility. *Nematology* **6**, 23–32 (2004).
22. M. K. Hasna, V. Insunza, J. Lagerlöf, B. Rämert, Food attraction and population growth of fungivorous nematodes with different fungi. *Ann. Appl. Biol.* **151**, 175–182 (2007).
23. K. A. Dekker *et al*., Novel lactone compounds from *Mortierella verticillata* that induce the human low density lipoprotein receptor gene: Fermentation, isolation, structural elucidation and biological activities. *J. Antibiot. (Tokyo)* **51**, 14–20 (1998).
24. N. Vandepol *et al*., Resolving the Mortierellaceae phylogeny through synthesis of multi-gene phylogenetics and phylogenomics. *Fungal Divers.* **104**, 267–289 (2020).
25. M. R. Boyd *et al*., Discovery of a novel antitumor benzolactone enamide class that selectively inhibits mammalian vacuolar-type (H^+)-atpases. *J. Pharmacol. Exp. Ther.* **297**, 114–120 (2001).
26. M. Pérez-Sayáns, J. M. Somoza-Martín, F. Barros-Angueira, J. M. Rey, A. García-García, V-ATPase inhibitors and implication in cancer treatment. *Cancer Treat. Rev.* **35**, 707–713 (2009).
27. S. P. Niehs *et al*., Mining symbionts of a spider-transmitted fungus illuminates uncharted biosynthetic pathways to cytotoxic benzolactones. *Angew. Chem. Int. Ed. Engl.* **59**, 7766–7771 (2020).
28. Y. Hayakawa *et al*., Oximidine III, a new antitumor antibiotic against transformed cells from *Pseudomonas* sp. II. Structure elucidation. *J. Antibiot. (Tokyo)* **56**, 905–908 (2003).
29. D. L. Galinis, T. C. McKee, L. K. Pannell, J. H. Cardellina, M. R. Boyd, Lobatamides A and B, novel cytotoxic macrolides from the tunicate *Aplidium lobatum*. *J. Org. Chem.* **62**, 8968–8969 (1997).
30. R. Ueoka *et al*., Genome mining of oxidation modules in trans-acyltransferase polyketide synthases reveals a culturable source for lobatamides. *Angew. Chem. Int. Ed. Engl.* **59**, 7761–7765 (2020).
31. Y. Sato *et al*., Detection of betaproteobacteria inside the mycelium of the fungus *Mortierella elongata*. *Microbes Environ.* **25**, 321–324 (2010).
32. Y. Takashima *et al*., Prevalence and intra-family phylogenetic divergence of *Burkholderiaceae*-related endobacteria associated with species of *Mortierella*. *Microbes Environ.* **33**, 417–427 (2018).
33. S. Ohshima *et al*., *Mycoavidus cysteinexigens* gen. nov., sp. nov., an endohyphal bacterium isolated from a soil isolate of the fungus *Mortierella elongata*. *Int. J. Syst. Evol. Microbiol.* **66**, 2052–2057 (2016).
34. Y. Guo *et al*., *Mycoavidus* sp. Strain B2-EB: Comparative genomics reveals minimal genomic features required by a cultivable *Burkholderiaceae*-related endofungal bacterium. *Appl. Environ. Microbiol.* **86**, e01018-20 (2020).
35. J. Uehling *et al*., Comparative genomics of *Mortierella elongata* and its bacterial endosymbiont *Mycoavidus cysteinexigens*. *Environ. Microbiol.* **19**, 2964–2983 (2017).
36. L. P. Partida-Martinez, C. Hertweck, Pathogenic fungus harbours endosymbiotic bacteria for toxin production. *Nature* **437**, 884–888 (2005).
37. G. Lackner, N. Moebius, C. Hertweck, Endofungal bacterium controls its host by an hrp type III secretion system. *ISME J.* **5**, 252–261 (2011).
38. C. Jain, L. M. Rodriguez-R, A. M. Phillippy, K. T. Konstantinidis, S. Aluru, High throughput ANI analysis of 90K prokaryotic genomes reveals clear species boundaries. *Nat. Commun.* **9**, 5114 (2018).
39. P. Yarza *et al*., Uniting the classification of cultured and uncultured bacteria and archaea using 16S rRNA gene sequences. *Nat. Rev. Microbiol.* **12**, 635–645 (2014).
40. K. Blin *et al*., antiSMASH 5.0: Updates to the secondary metabolite genome mining pipeline. *Nucleic Acids Res.* **47** (W1), W81–W87 (2019).
41. K. Scherlach, L. P. Partida-Martinez, H. M. Dahse, C. Hertweck, Antimitotic rhizoxin derivatives from a cultured bacterial endosymbiont of the rice pathogenic fungus Rhizopus microsporus. *J. Am. Chem. Soc.* **128**, 11529–11536 (2006).
42. L. P. Partida-Martinez *et al*., Rhizonin, the first mycotoxin isolated from the zygomycota, is not a fungal metabolite but is produced by bacterial endosymbionts. *Appl. Environ. Microbiol.* **73**, 793–797 (2007).
43. G. Lackner, L. P. Partida-Martinez, C. Hertweck, Endofungal bacteria as producers of mycotoxins. *Trends Microbiol.* **17**, 570–576 (2009).
44. C. Almeida *et al*., Unveiling concealed functions of endosymbiotic bacteria harbored in the ascomycete stachylidium bicolor. *Appl. Environ. Microbiol.* **84**, e00660-18 (2018).
45. K. Scherlach, B. Busch, G. Lackner, U. Paszkowski, C. Hertweck, Symbiotic cooperation in the biosynthesis of a phytotoxin. *Angew. Chem. Int. Ed. Engl.* **51**, 9615–9618 (2012).
46. S. Geisen *et al*., The soil food web revisited: Diverse and widespread mycophagous soil protists. *Soil Biol. Biochem.* **94**, 10–18 (2016).
47. J. van den Hoogen *et al*., A global database of soil nematode abundance and functional group composition. *Sci. Data* **7**, 103 (2020).
48. M. P. Smith *et al*., A liquid-based method for the assessment of bacterial pathogenicity using the nematode *Caenorhabditis elegans*. *FEMS Microbiol. Lett.* **210**, 181–185 (2002).
49. E. J. Ragsdale, J. Crum, M. H. Ellisman, J. G. Baldwin, Three-dimensional reconstruction of the stomatostylet and anterior epidermis in the nematode *Aphelenchus avenae* (Nematoda: Aphelenchidae) with implications for the evolution of plant parasitism. *J. Morphol.* **269**, 1181–1196 (2008).
50. S. S. Schmieder *et al*., Bidirectional propagation of signals and nutrients in fungal networks via specialized hyphae. *Curr. Biol.* **29**, 217–228.e4 (2019).
51. G. W. Yeates, T. Bongers, R. G. De Goede, D. W. Freckman, S. S. Georgieva, Feeding habits in soil nematode families and genera-an outline for soil ecologists. *J. Nematol.* **25**, 315–331 (1993).
52. A. Tayyrov, S. S. Schmieder, S. Bleuler-Martinez, D. F. Plaza, M. Künzler, Toxicity of potential fungal defense proteins towards the fungivorous nematodes *Aphelenchus avenae* and *Bursaphelenchus okinawaensis*. *Appl. Environ. Microbiol.* **84**, e02051-18 (2018).
53. R. Hermenau *et al*., Gramibactin is a bacterial siderophore with a diazeniumdiolate ligand system. *Nat. Chem. Biol.* **14**, 841–843 (2018).
54. R. R. Wick, L. M. Judd, C. L. Gorrie, K. E. Holt, Completing bacterial genome assemblies with multiplex MinION sequencing. *Microb. Genom.* **3**, e000132 (2017).
55. S. Blanton *et al*., A web-based carepartner-integrated rehabilitation program for persons with stroke: Study protocol for a pilot randomized controlled trial. *Pilot Feasibility Stud.* **5**, 58 (2019).
56. S. Bleuler-Martínez *et al*., A lectin-mediated resistance of higher fungi against predators and parasites. *Mol. Ecol.* **20**, 3056–3070 (2011).

PNAS
www.pnas.org

Supplementary Information for

Bacterial endosymbionts protect beneficial soil fungus from nematode attack

Hannah Büttner,[†,1] Sarah P. Niehs,[†,1] Koen Vandelannoote,[2] Zoltán Cseresnyés,[3] Benjamin Dose,[1] Ingrid Richter,[1] Ruman Gerst,[3,4] Marc Thilo Figge,[3,5] Sacha J. Pidot,[*,2] Christian Hertweck[*,1,4]

Corresponding authors: Christian Hertweck, Sacha J. Pidot
* To whom correspondence may be addressed. Email: christian.hertweck@leibniz-hki.de; sacha.pidot@unimelb.edu.au

This PDF file includes:

Other supplementary materials for this manuscript include the following:

Experimental Procedures

Bacterial and fungal strains

Strains of this study are listed in Table S 1. All media used in this study are listed in Table S 2. Sterilization of the media took place at 120 °C for 20 min.
Burkholderia sp. strain B8 (HKI-0404; syn. *Mycetohabitans*) (1) was isolated from *Rhizopus microsporus* Tieghem var. *microsporus* CBS 308.87 by subsequent procedure: The fungus was inoculated into MGY+M9 medium and grown at 30 °C and 110 rpm until increasing turbidity of the medium was observed by eye. The culture was centrifuged (12,000 × *g*, 10 min, 25 °C). A small volume was separated from the upper surface of the culture and spread on NAG agar. After growth at 30 °C for 3–4 days, bacterial colonies were inoculated into MGY+M9 medium and slowly upscaled.
Fungal spores were stored in 50 % glycerol at –20 °C for long-term storage. For short-term storage they were grown on PDA and kept at 4 °C or room temperature.

Table S 1. Strains used in this study.

Strain	No.	Original site of isolation
Burkholderia sp.	HKI0404, strain B8	CBS 308.87 in this study
Rhizopus microsporus Tieghem var. *microsporus*	CBS 308.87 (NRRL 28628)	Human necrotic tissue after a spider bite, Australia
Candidatus Mycoavidus necroximicus	-	*Mortierella verticillata* NRRL 6337
Mortierella verticillata	NRRL 6337 (CBS 131.66)	Sandy forest soil, United Kingdom
Mortierella verticillata	NRRL 6369 (CBS 100561)	Soil of Great Bear Lake, Canada
Mortierella verticillata	SF9852 (CBS 346.66)	Tundra soil, Alaska
Mortierella verticillata	SF9853 (CBS 220.58)	Soil under *Betula* sp., France
Mortierella verticillata	SF9854 (CBS 225.35)	Former West Germany
Mortierella verticillata	SF9856 (CBS 315.52)	Forest soil, former West Germany
Escherichia coli	OP50	-
Caenorhabditis elegans	Wild-type N2 (var. Bristol)	-
Aphelenchus avenae	Bastian, 1865	-

Table S 2. Media used in this study.

Media	Composition (L^{-1})
MGY+M9 medium	10 g Glycerol, 1.25 g yeast extract (autolyzed yeast cells, BD, Bacto), 960 mL water, sterilization, add: 20 mL M9 salt A, 20 mL M9 salt B
MM9 medium	2 g Amino acid mix, 10 g glycerol, 900 mL water, sterilization, add: appropriate antibiotics, 20 mL M9 salt A, 20 mL M9 salt B, 16.8 mL L-leucine solution (100 mM), 5 mL L-histidine solution (60 mM), each 10 mL of L-lysine (100 mM), L-tryptophan (40 mM), L-methionine solution (40 mM), 2 mL vitamin solution, 1 mL trace element solution
PDB/PDA	Potato dextrose broth or agar (BD, Bacto), sterilization
NAG agar	Standard nutrient agar I (Merck), 10 g glycerol, sterilization
Modified medium 2	3 % Glycerol, 1 % glucose, 0.5 % peptone, 0.2 % NaCl, pH 6.0, sterilization
LB	Lysogeny broth (BD, Bacto), sterilization
TSB	Tryptone soy broth (BD, Bacto), sterilization
CYE	Charcoal yeast extract medium; 10 g yeast extract (autolyzed yeast cells, BD, Bacto), 10 g ACES, 1 g potassium oxoglutamate, 2 g active charcoal, pH 6.9, sterilization, add: 0.25 g Fe-pyrophosphate (sterile filtered)
K-medium	3.1 g NaCl, 2.4 g KCl, sterilization
NGM	3 g NaCl, 2.5 g peptone (BD, Bacto), 17 g agar, sterilization, add (sterile): 5 mg cholesterol, 0.11 g $CaCl_2$, 0.25 g $MgSO_4$, 2.7 g KH_2PO_4, 0.89 g K_2HPO_4
M9 salts A	350 g K_2HPO_4, 100 g KH_2PO_4, sterilization
M9 salts B	29.4 g Sodium citrate, 50 g $(NH_4)_2SO_4$, 5 g $MgSO_4$, sterilization
Amino acid mix	L-Amino acids in equal amounts: alanine, asparagine, cysteine, glutamate, isoleucine, serine, arginine, aspartate, glutamine, glycine, proline, threonine, valine
Vitamin solution	10 mg Folic acid, 6 mg biotin, 200 mg *p*-aminobenzoic acid, 1 g thiamine-HCl, 1.2 g pantothenic acid, 1 g riboflavin, 2.3 g nicotinic acid, 12 g pyridoxine HCl, 100 mg vitamin B_{12}
Trace element solution	40 mg $ZnCl_2$, 200 mg $FeCl_3 \times 6\ H_2O$, 10 mg $CuCl_2 \times 2\ H_2O$, 10 mg $MnCl_2 \times 4\ H_2O$, 10 mg $Na_2B_4O_7 \times 10\ H_2O$, 10 mg $(NH_4)_6Mo_7O_{24} \times 4\ H_2O$

Identification of endosymbionts in *Mortierella verticillata*

Amplification of bacterial 16S rDNA and phylogenetic analysis. Fungal strains were cultivated in MM9 medium at 26 °C and orbital shaking at 110 rpm. The genomic DNA was isolated either from the turbid supernatant or the disrupted fungus itself with the MasterPure DNA Purification Kit (Epicentre). 16S rDNA was amplified using gDNA, the primers 8F (AGA GTT TGA TCC TGG CTC AG) and 1492R (CGG TTA CCT TGT TAC GAC TT) with Phusion High-Fidelity PCR Master Mix with HF Buffer (NEB). PCR program: 30 cycles of 95 °C for 15 s, 65 °C for 15 s, 72 °C for 1 min 40 s.

The 16S rDNA sequences of the *Mortierella* endosymbionts
were uploaded to the NCBI database:
BRE_MvertCBS_346.66 MZ330684;
BRE_MvertCBS_220.58 MZ330685;
BRE_MvertCBS_225.35 MZ330686;
BRE_MvertCBS_315.52 MZ330687;
BRE_MvertCBS_100561 MZ330688.

Phylogenetic analysis of 16S rDNA of *Mortierella* endosymbionts (and further representatives of the Burkholderiaceae family) were performed as follows: The sequence alignment was performed using Clustal Omega with default settings (2). Maximum likelihood phylogeny was constructed using IQ-tree 2. Ultrafast bootstrapping (1,000 iterations) analysis was performed (3). Bootstrap values >80 % are shown at nodes. *Wolbachia pipientis* (16S rDNA gene sequence accession number: AY833061.1) was used as outgroup. Sequences where retrieved from the NCBI database, grouping of sequences to clades was performed in accordance to (4) (Figure S 3).

Preparation of aposymbiotic fungal strains. Fungal strains were continuously cultivated at 24 °C on PDA plates (Bacto, BD) containing 40 µg mL^{-1} ciprofloxacin or 50 µg mL^{-1} kanamycin for 4 months. After phenotypical changes were observed by eye, the *M. verticillata* NRRL 6337 fungal cultures were extracted with 1:1 volume of ethyl acetate and checked for production of **3** and **4**. Amplification of bacterial 16S rDNA from cured fungi did not lead to any amplicon (Figure S 2). Final examination of the cured fungal strains occurred by fluorescent staining.

Fluorescent staining. Fungal strains (wild type and aposymbiotic strains) were visualized by a Zeiss LSM 710 confocal laser-scanning microscope. First, fungal hyphae were stained with Calcofluor White Stain (Sigma) and SYTO 9 Green (Invitrogen) for 5 min, then washed in 0.85 % NaCl solution, and checked for endosymbionts. Wavelengths were adjusted as specified by the manufacturer (Figures S 1–S 2).

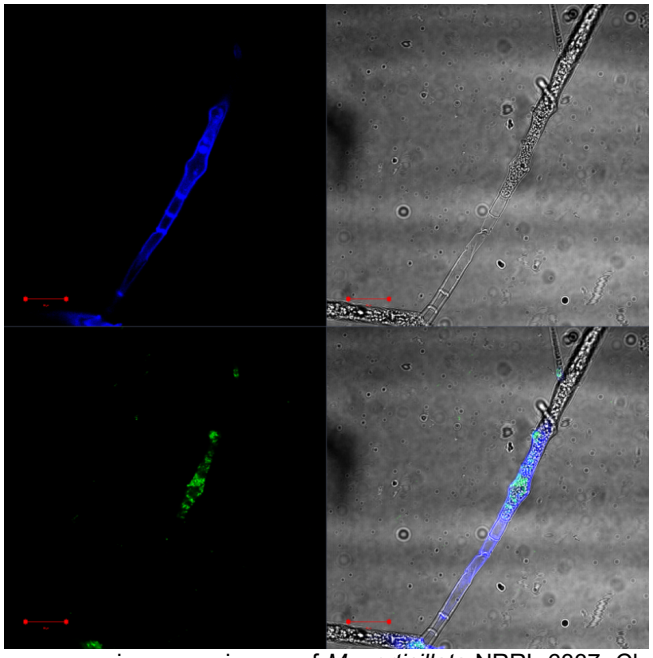


Figure S 1. Fluorescence microscopy image of *M. verticillata* NRRL 6337. Channel 1 Calcofluor White staining (to stain the fungal cell wall, blue; top left), channel 2 bright field (top right), channel 3 SYTO 9 Green (to stain nucleic acids and visualize bacteria, green; bottom left), channel 4 overlay (bottom right). Scale matches 20 µm.

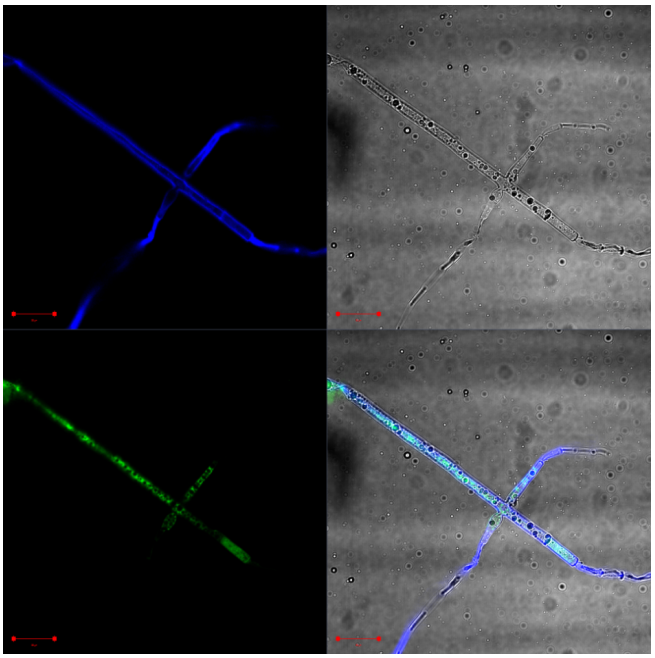


Figure S 2. Fluorescence microscopy image of cured (aposymbiotic) *M. verticillata* NRRL 6337. Channel 1 Calcofluor White staining (to stain the fungal cell wall, blue; top left), channel 2 bright

field (top right), channel 3 SYTO 9 Green (to stain nucleic acids and visualize bacteria, green; bottom left) (overexposure), channel 4 overlay (bottom right). Scale matches 20 µm.

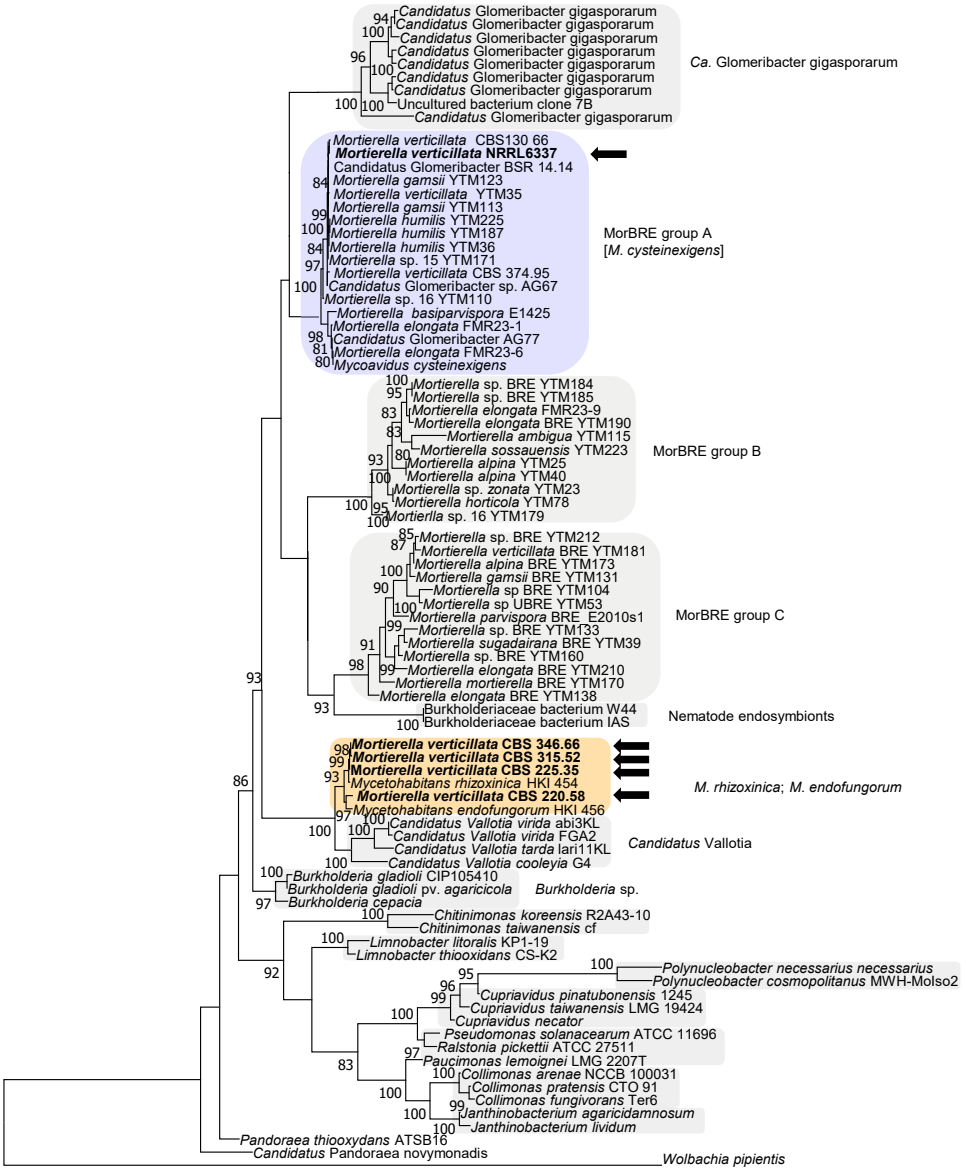


Figure S 3 Phylogenetic analysis of amplified 16S rDNA from *Mortierella* symbionts. Strains of this study are marked with an arrow. Additional bacterial sequences labelled "*Mortierella*" were extracted from a previous publication (4). Abbreviation: BRE *Mortierella*, *Burkholderia*-related endosymbiont of *Mortierella* spp.

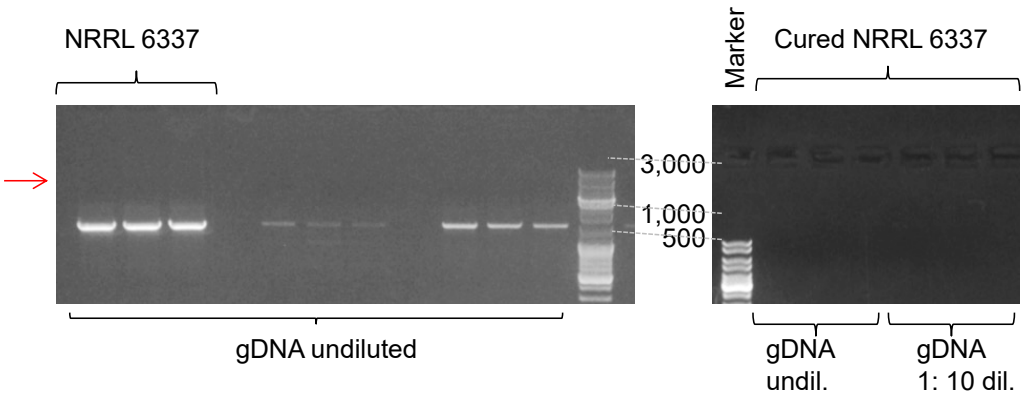


Figure S 4 Amplification of bacterial 16S rDNA from symbiotic and cured *M. verticillata* NRRL 6337 strains using the primers 8F (AGA GTT TGA TCC TGG CTC AG) and 1492R (CGG TTA CCT TGT TAC GAC TT**)**. Marker in Da. Undil, not diluted. Marker sizes in Da.

Endosymbiont isolation attempts (for *M. verticillata* spp.). In short, isolation attempts include bacteria derived from turbid supernatant of fungal cultures or from disrupted hyphae (homogenizer or bead mill). The bacterial mixture was filtered through a 40 µm sieve (Corning Cell Strainer) and a 5 µm filter (Cameo, Roth) before inoculation. Tested media included: MGY+M9, MM9, PDB, NAG, LB, TSB, CYE medium (5), as well as CYE medium containing autoclaved or sterile filtered *M. verticillata* cultures grown in PDB. The same media were also tested with additional L-cysteine, amino acid mix solution (Table S 2) and 3 % glycerol. Every assay was performed with liquid medium and/or on an agar plate. If fungal growth occurred on plates, these parts were cut out with a sterile scalpel. For liquid media also the addition of amphotericin in concentrations of 1 µM, 5 µM or 10 µM was tested to inhibit fungal growth. Cultures were incubated at 20 °C, 26 °C, 30 °C and 37 °C for up to 30 days. Additionally, the infection of a cured fungal strain with bacterial filtrate did not result in living endosymbionts.

Genome assembly for *Candidatus* Mycoavidus necroximicus

M. verticillata NRRL 6337 was grown in 3 L MM9 medium (Table S 2) with orbital shaking at 160 rpm and 26 °C. Mild physical sheering through shaking culture in baffled flasks led to increasing turbidity of the culture supernatant over time. Microscopic analysis revealed a high number of bacteria in comparison to mycelia in the turbid medium. The cultures were controlled for turbidity by eye every day. If the supernatant turned significantly turbid, the culture was twice filtered through a membrane (pore diameter 40 µm; Corning cell strainer) and centrifuged (12,000 × *g*, 25 °C, 10 min) until a stabile pellet occurred. The genomic DNA was extracted according to manufacturer's recommendations with the MasterPure DNA Purification Kit (Epicentre).
The extracted gDNA from *M. verticillata* was prepared for sequencing on both the Oxford Nanopore MinION and Illumina NextSeq platforms. For long-read sequencing on the MinION platform, DNA quality was evaluated by pulsed-field gel electrophoresis and prepared for sequencing according to the protocol of the Ligation Sequencing kit (Oxford Nanopore). DNA was loaded onto a single MinION flow cell and data was collected over a 72 hour period. DNA was prepared for sequencing on the Illumina NextSeq platform using the Nextera XT DNA preparation kit (Illumina) with × 150 bp paired end chemistry and with a targeted sequencing depth of >50 ×. Combined MINion and Illumina sequencing data were assembled using the Unicycler hybrid assembler (6, 7), following which a single contig 2.2 Mb containing a 98.82 % match to the *Mycoavidus cysteinexigens* 16S rDNA gene was extracted and evaluated for secondary metabolite loci using antiSMASH version 5 (8).

Identification and annotation of secondary metabolite gene cluster

Candidatus Mycoavidus necroximicus differed from other so far characterized *Mycoavidus* species. In contrast, we identified many similarities in the natural product arsenal between the *Mortierella* symbiont *Candidatus* Mycoavidus necroximicus and symbionts of *R. microsporus* along with their overall high potential for secondary metabolism. In addition to the necroximes (**3** and **4**), also the lasso peptide mycetohabin-15 (**8**) (35), originally reported from the fungal endosymbiont *Mycetohabitans* (previously *Paraburkholderia*) *rhizoxinica*, was detected from cultures of *Candidatus* Mycoavidus necroximicus (Figure S 5). Furthermore, through intensified bioinformatics analyses all genes necessary for F420 biosynthesis were identified, an important redox co-factor (9-11) that has been described in endofungal *Burkholderia* sp. and other bacteria before. For the detection of secondary metabolites antiSMASH (8) and PKS/NRPS analysis (12) were used for annotation of the biosynthetic gene cluster and the necroxime cluster (Figure S 6). Available genome assemblies of *Mycoavidus cysteinexigens* AG77 (13), B2-EB (14) and B1-EB^T (15) were processed in the same way. In addition, precursor sequences coding for lasso peptide assembly were identified manually (putative core peptides are highlighted in bold; putative cyclization sites in red):

a) *Candidatus* Mycoavidus necroximicus

MTKSKAINTQEIQLDDDALMEFCASESTM	**GAVGEKNEAGFGKYDDDAV**
MIKNQELNSQAIQLDDEALTQFSASEATM	**GGSGQYREAGVGRFL*** (Mycetohabin-15)
MTDSKKTTTQDTQLKDEALTEFCASESTM	**GGSGQYREAGVGRFL*** (Mycetohabin-15)

b) *M. cysteinexigens* AG77

MIKNQELNQDIQLDDEALTQFCASEATM	**GGSGQYKEAGVGRFL***
MIKNQELNQDIQLDDEVLTQFCASEATM	**GGSGKYKEAGVGRFL***
MTNSKEIKIQETQLQDETLSEFCASKATM	**GGSGQYREAGVGRFL** (Mycetohabin-15)
MTKSKELSQDIQLEETLMEFCASEATM	**GAVGEKNEAGFGKY**

c) *M. cysteinexigens* B1-EB^T

MIKNQELNQDIQLDDEVLTQICASEATM	**GGSGQYREAGVGRFL*** (Mycetohabin-15)
MIKNQELNQDIQLDDEVLTQFCASEATM	**GGSGKYKEAGVGRFL**
MTKSKELSQDIQLEETLMEFCASEATM	**GAVGEKNEAGFGKY**
MTNSKENKIQEIQLQDETLSEFCASEATM	**GGSGQYREAGVGRFL*** (Mycetohabin-15)

d) *M. cysteinexigens* B2-EB^T

MTNSKETKTQETQLQDETLSEFCASEATM	**GGSGQYREAGVGRFL** (Mycetohabin-15)

* 1–2 copies of the core peptide with different leader peptides detected in the genome (only assigned as one lassopeptide)

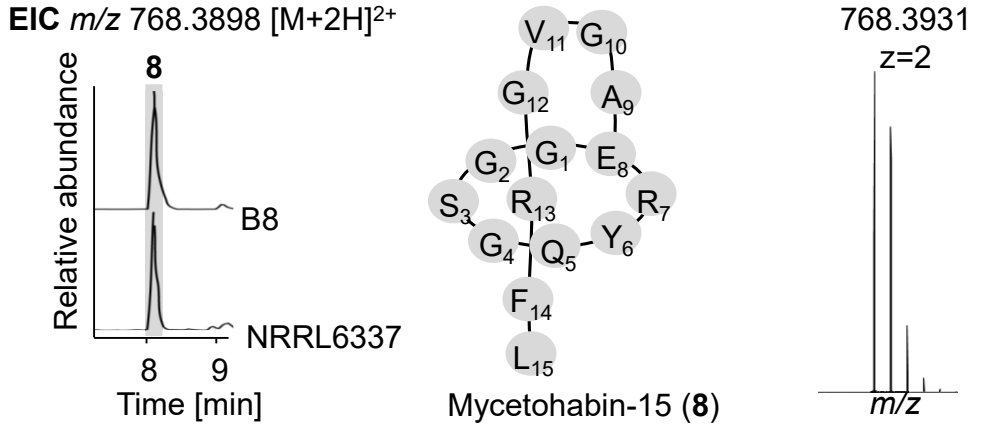


Figure S 5. Mycetohabin-15. Extracted ion chromatograms of mycetohabin-15 (**8**) (*m/z* 768.3930 [*M*+2H]$^{2+}$) observed in extracts of *Candidatus* Mycoavidus necroximicus (NRRL 6337) and *Burkholderia* sp. strain B8. Structure of the lasso peptide **8** and isotopic pattern.

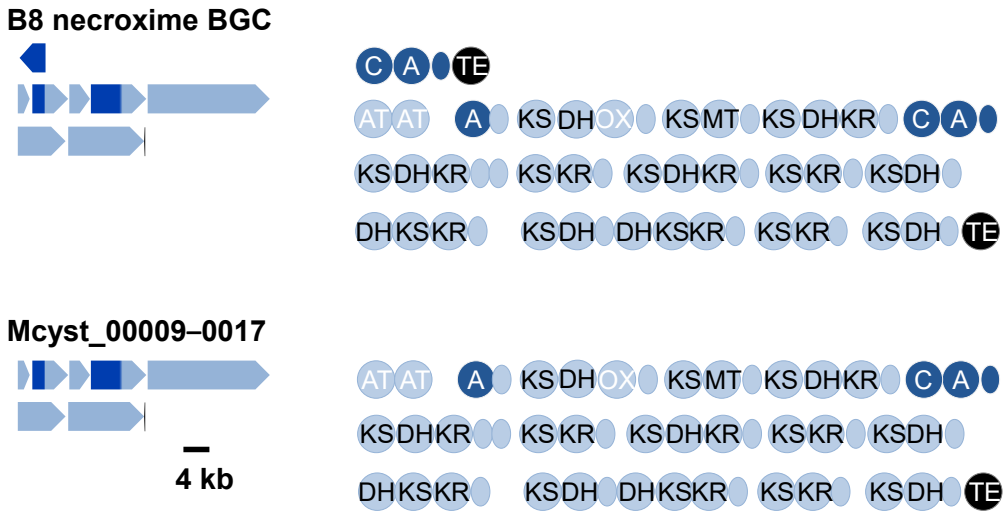


Figure S 6. Architectures of necroxime biosynthetic assembly lines in *Burkholderia* sp. strain B8 (top) and *Candidatus* Mycoavidus necroximicus (below).

Table S 3. Proteins encoded in the necroxime assembly line of *Candidatus* Mycoavidus necroximicus and putative up- and downstream proteins. Id./Sim. – Identity/Similarity.

Locus Mcyst_ 00	[bp]	Putative protein	SwissProt/ PDB entry	Acces-sion #	Organism	Id./ Sim.
024	1,191	Sodium/hydrogen exchanger	UPF0391 membrane protein XCV1406	Q3BVS6.1	*Xanthomonas campestris* pv. *vesicatoria* str. 85-10	35%/ 62%
023	1,116	Putative glutamate-cysteine ligase 2	Gamma-glutamyl-cysteine synthetase 2	B2T7N6.1	*Paraburk-holderia phytofirmans* PsJN	73%/ 86%
022	447	Hypothetical protein	Histone-lysine N-methyltrans-ferase 2A	Q03164.5	*Homo sapiens*	40%/ 55%
021	603	Hypothetical protein	Uncharacterized protein y4rO	P55648.1	*Sinorhizobium fredii* NGR234	33%/ 58%
020	876	IS982 family tranposase	Putative transposase	Q08082.2	*Brucella ovis* ATCC 25840	28%/ 44%
019	219	Hypothetical protein	-	-	-	-
018	190	Hypothetical protein	-	-	-	-
017	1,899	Malonyl CoA-acyl carrier protein transacylase	AT	A7Z4X8.1	*Bacillus velezensis* FZB42	48%/ 66%
016	5,541	PKS synthase/ NRPS synthetase	PKS	P40806.3	*Bacillus subtilis* subsp. *subtilis* str. 168	36%/ 51%
015	3,177	Polyketide synthase	PKS	P40806.3	*Bacillus subtilis* subsp. *subtilis* str. 168	33%/ 52%
014	8,541	PKS/NRPS	Polyketide synthase PksN	O31782.3	*Bacillus subtilis* subsp. *subtilis* str. 168	30%/ 47%
013	18,807	Polyketide synthase	PKS	P40806.3	*Bacillus subtilis* subsp. *subtilis* str. 168	31%/ 49%

Table S 3 continued. Proteins encoded in the necroxime assembly line of *Candidatus* Mycoavidus necroximicus and putative up- and downstream proteins. Id./Sim. – Identity/Similarity.

Locus Mcyst_ 00	[bp]	Putative protein	SwissProt/ PDB entry	Acces- sion #	Organism	Id./ Sim.
012	7,329	Polyketide synthase	PKS	P40806.3	*Bacillus subtilis* subsp. *subtilis* str. 168	43%/ 57%
011	11,652	Polyketide synthase	Polyketide synthase PksN	O31782.3	*Bacillus subtilis* subsp. *subtilis* str. 168	47%/ 63%
010	183	Hypothetical protein	Rho guanine nucleotide exchange factor 11	Q9ES67. 1	*Rattus norvegicus*	38%/ 47%
009	1,416	Epi-isozizaene 5-monooxygenase	Probable cytochrome P450 311a1	Q9VYQ7. 1	*Drosophila melanogaster*	28%/ 43%
008	237	Hypothetical protein	Glutamyl-tRNA(Gln) amidotrans- ferase subunit B-1	C1MIE8.1	*Micromonas pusilla* CCMP1545	32%/ 44%
007	1,665	Lanthionine synthetase C family protein	-	-	-	-
006	132	Hypothetical protein	-	-	-	-
005	1,038	Hypothetical protein	3-keto-5-aminohex- anoate cleavage enzyme	B0VHH0. 1	*Candidatus* Cloacimonas acidaminovorans str. Evry	25%/ 43%

Comparative genomics

Genome sequences of *M. cysteinexigens* B1-EB (NCBI accession AP018150.1), B2-EB (NCBI accession AP021872.1) and *M. cysteinexigens* AG77 (PATRIC accession 224135.3) were used for comparative genome analyses. The *M. cysteinexigens* AG77 genome was reorientated to start with *dnaA* (as for the other *Mycoavidus* genomes) and reannotated with Prokka 1.14.5.(16) Comparison of the three *Mycoavidus* genomes was performed using Roary v3.12.0 (17) with an 70 % protein identity cutoff. Genome synteny figures were constructed using Circos (18).

In general, evidence for the high dependence of these *Mycoavidus* symbionts on their hosts can be observed in the genomes. *M. cysteinexigens* AG77 has previously been found to be missing the biosynthetic pathway for the amino acid cysteine (19). Investigation of biosynthetic pathways in strain *Ca.* Mycoavidus necroximicus showed the loss of further amino acid biosynthesis pathways, including those for cysteine, histidine, isoleucine, leucine, methionine, threonine, tryptophan and tyrosine (https://papers.genomics.lbl.gov/cgi-bin/gapView.cgi) (20). This suggests that *Candidatus* Mycoavidus necroximicus relies very heavily on *M. verticillata* for the production of amino acid building blocks and might also explain why the fungus grows more aerial hyphae in the absence of endosymbionts. This is further reflected in the inability to grow *Candidatus* Mycoavidus necroximicus in an aposymbiotic manner, while axenic growth of *M. cysteinexigens* B1-EB^T, B2-EB and AG77 can be achieved by supplementing growth media with cysteine (5).

General analytical methods

Analytical HR-ESI-LC/MS. Exactive Orbitrap High Performance Benchtop LC/MS (Thermo Fisher Scientific) with an electron spray ion source and an Accela HPLC System, C18 column (Betasil C18, 150 × 2.1 mm, Thermo Fisher Scientific), solvents: acetonitrile and water (both supplemented with 0.1 % formic acid), flow rate: 0.2 mL min^{-1}; program: hold 1 min at 5 % acetonitrile, 1–16 min 5–98 % acetonitrile, hold 3 min 98 % acetonitrile, 19–20 min 98 % to 5 % acetonitrile, hold 3 min at 5 % acetonitrile.

MS/MS (tandem mass spectrometry). QExactive Orbitrap High Performance Benchtop LC/MS (ThermoFisher) with an electron spray ion source and an Accela HPLC System, C18 column (Accucore C18 2.6 µm, 100 × 2.1 mm, Thermo Fisher Scientific) and the following solvent system: acetonitrile and water (both supplemented with 0.1 % formic acid) at a flow rate of 0.2 mL min^{-1}; gradient: 0–10 min 5–98 % acetonitrile, hold 4 min 98 % acetonitrile, 14–14.1 min 98 % to 5 % acetonitrile, hold 6 min at 5 % acetonitrile.

NMR. 600 MHz Avance III Ultra Shield (Bruker) and signals were referenced to the residual solvent signal. ^{1}H 600 MHz, ^{13}C 150 MHz; NMR solvent: DMSO-d_6.

Optical rotation. Jasco P-1020 polarimeter, Na light (589 nm), at 25 °C, 50 mm cell length, c 2 w/v%, dissolved in 83% acetonitrile (83% MeCN).

Identification, extraction and isolation of secondary metabolites

Extraction of necroxime D (CJ-12,290) (4) and necroxime C (CJ-13,357) (3). *M. verticillata* NRRL 6337 was either cultivated in modified medium 2 (21) for 7 days, at 160 rpm and 26 °C or on PDA plates at 26 °C for 28 days (Table S 2). The cultures were extracted with 1:1 volume of ethyl acetate overnight. Following, the organic phase was concentrated under reduced pressure and the residue was dissolved in a small volume of methanol. The extracts were measured *via* LC/MS.
Measured *m/z* 459.1833 and 459.1758 $[M+H]^+$, calculated 459.1762, $C_{23}H_{27}N_2O_8$ (CJ-12,290/CJ-13,357; **4**/**3**)

Extraction of lassopeptide mycetohabin-15 (8). *M. verticillata* NRRL 6337 or *Burkholderia* sp. HKI-404 (strain B8) were cultivated in MM9 medium for 7 days, with orbital shaking at 160 rpm and 26 °C. The absorber resin XAD-2 was added for 30 min to the culture, followed by separation of the resin and extraction with 100 % methanol overnight. The organic phase was concentrated under reduced pressure and the residue was dissolved in a small volume of methanol. The extracts were measured *via* LC/MS.
Measured *m/z* 768.3931 $[M+2H]^{2+}$ (Mycetohabin-15; **8**)

Production of necroxime A (1). *Burkholderia* sp. strain B8 wild type or Δ*necA* mutant were cultivated in MGY+M9 medium for 4–5 days with orbital shaking at 110 rpm and 30 °C. Absorber resin XAD-2 was added to the bacterial culture for 30 min, separated from the culture and extracted in 100 % methanol for 4 hours. The organic phase was concentrated under reduced pressure and the residue was dissolved in a small volume of methanol. The extracts were measured *via* LC/MS.
Measured *m/z* 673.3088 $[M+H]^+$, calculated 673.3079, $C_{33}H_{45}N_4O_{11}$ (necroxime A; **1**)

Isolation of necroxime D (4) from *M. verticillata* NRRL 6337. Necroxime D (**4**) was the main metabolite detected in fresh extracts of the *Mycoavidus* endosymbionts with no prior light-exposure; as **4** then easily isomerizes into **3** under long-term light exposure (21), we isolated only species **4** (Figure S 7). The fungal host *M. verticillata* NRRL 6337 was cultivated on PDA plates at 26 °C for 28 days. The culture was extracted twice with 1:1 volume of ethyl acetate overnight. Following, the organic phase was concentrated under reduced pressure and the residue was dissolved in a small volume of methanol. The extract was pre-fractionated on an open Sephadex LH-20-column with methanol and the necroxime-containing fraction was further purified with a preparative HPLC under following conditions: A: H_2O + 0.01 % TFA, B: Methanol; 15–100 % B in 35 min, 15 mL min^{-1} (Phenomenex, Luna, 10 µm, C18(2), 100 Å, 250 × 21.2 mm).

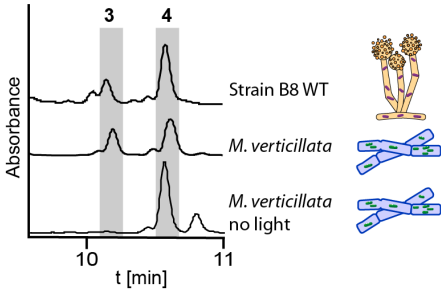


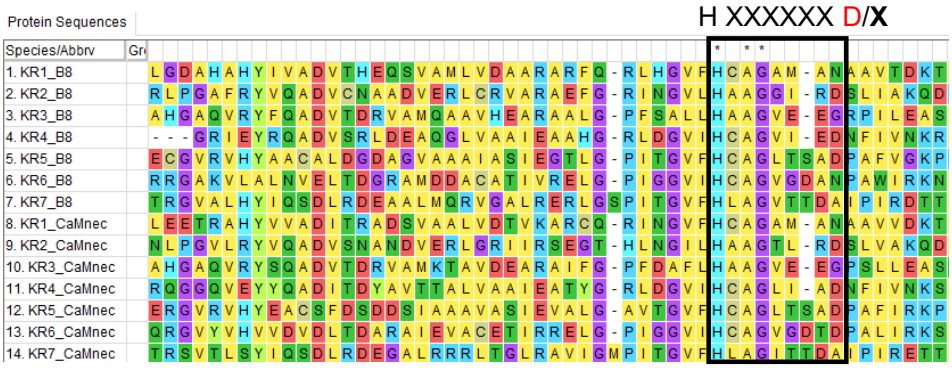
Figure S 7. Metabolic profiles of necroxime-containing extracts from *Burkholderia* sp. strain B8 and *M. verticillata* (cultivated and extracted under normal and light-reduced conditions).

Comparison of necroxime C and D (3 and 4) from *Burkholderia* sp. strain B8 and *M. verticillata* NRRL 6337. For the identification of the necroximes C and D from *M. verticillata* NRRL 6337 the previously described and isolated substances from *Burkholderia* sp. strain B8 were used as authentic standards (22). Co-injection revealed the same retention times of the substances isolated from the two producers. MS/MS fragmentation showed identical fragmentation patterns of the substances (Figure S 13–16). Comparison of the chemical shifts in NMR measurements of necroximes C and D (**3** and **4**) from *Burkholderia* sp. strain B8 and necroxime D (**4**) from *M. verticillata* revealed the identical configuration within the respective molecules (Table S 9). To unambiguously verify that the isolated necroximes from *Burkholderia* sp. strain B8 and *M. verticillata* NRRL 6337 are identical molecules and no enantiomers, we determined the optical rotation of necroxime C and D (**3** and **4**) isolated from *Burkholderia* sp. strain B8. The comparison of the measured values with the published values for necroxime C (CJ-13,357) (**3**) and necroxime D (CJ-12,290) (**4**) (21) identified the molecules to be the same, as a similar rotation can be measured (Table S 4). Additionally, we performed genome mining with the two BGCs from necroximes and CJ-compounds, extracted the ketoreductase domain sequences that determine the absolute configurations of the incorporated OH residues, and compared the specificity codes using Mega7 (23) and ClustalW (24). HXXXXXXD codes for D-β-OH as shown by Caffrey (25). That way we could determine the absolute configuration of the OH groups and, consequently, show that necroximes C–D are stereochemically identical to CJ-13,357 and CJ-12,950 (Figure S 8).

Table S 4. Comparison of optical rotation of necroxime C and D (3 and 4) isolated from *Burkholderia* sp. strain B8 and *M. verticillata* NRRL 6337.

	necroxime C (CJ-13,357) (**3**)	necroxime D (CJ-12,290) (**4**)
M. verticillata NRRL 6337 [α]D (25 °C, MeOH)	+ 107.8° (c 0.23)	+ 99.5° (c 0.22)
Burkholderia sp. strain B8 [α]D (25 °C, MeOH)	+ 89.9° (c 0.12)	+ 74.4° (c 0.25)

A



B

Organism	Module nr.	KR specificity code	Prediction
Mycetohabitans sp. strain B8	3	H CAGAMA **N**	L
Ca. M. necroximicus	3	H CAGAMA **N**	L
Mycetohabitans sp. strain B8	4	H AAGGIR **D**	**D**
Ca. M. necroximicus	4	H AAGTLR **D**	**D**
Mycetohabitans sp. strain B8	5	H AAGVEE **G**	L
Ca. M. necroximicus	5	H AAGVEE **G**	L
Mycetohabitans sp. strain B8	6	H AAGVIE **D**	**D**
Ca. M. necroximicus	6	H AAGLIA **D**	**D**
Mycetohabitans sp. strain B8	7	H CAGLTS **A**	L
Ca. M. necroximicus	7	H CAGLTS **A**	L
Mycetohabitans sp. strain B8	9	H CAGVGD **A**	L
Ca. M. necroximicus	9	H CAGVGD **T**	L
Mycetohabitans sp. strain B8	12	H LAGVTT **D**	**D**
Ca. M. necroximicus	12	H LAGITT **D**	**D**

Figure S 8. Multiple Sequence Alignment using KR domain sequences encoded in the necroxime BGCs of *Mycetohabitans* sp. strain B8 and *Ca.* Mycoavidus necroximicus unveils predicted configurations of OH groups in the natural products. A) Part of the alignment showing specificity code HXXXXXXD/X. B) Extracted specificity codes according to module architecture. Comparison of KR domain specificities extracted from strain B8 and *Ca.* Mycoavius necroximicus reveals identical L/D distibution.

Nematodal strains and cultivation conditions

The model organism *Caenorhabditis elegans* (*C. elegans* Genetics Centre (CGC, University of Minnesota, USA)) was maintained on *Escherichia coli* OP50 seeded on NGM agar (26). For culture maintenance a small, nematode-containing agar-piece of NGM was transferred onto fresh *E. coli*-covered NGM plates. Plates were kept at 20 °C for 4–7 days. *E. coli* OP50 was cultured in LB medium. If prepared for an assay, nematodes were washed from the plates with K-medium and left for settling at 4 °C for 30 min. Supernatant was discarded and resuspended in 8 mL K-medium for usage in experiments.

The nematode *Aphelenchus avenae* was obtained as a kind gift from Prof. Dr. Markus Künzler (ETH Zürich) and kept on a sporulation-deficient *Botrytis cinerea* strain (BC-3), grown on malt extract agar (MEA) containing 100 µg mL^{-1} chloramphenicol at 21 °C. Harvest of nematodes was performed via Baermann funneling as described earlier, with small variations (27). Nematode-containing plates were cut into small pieces and left upside-down overnight in a funnel lined with miracloth (Merck) and filled with K-medium. After release of the funnel, nematodes were sterilized for two hours in K-medium containing 100 mM geneticin (G418) and 25 µg mL^{-1} kanamycin, washed once with K-medium and plated onto 1.5 % agar plates containing 200 mM geneticin and 50 µg mL^{-1} kanamycin. After 24–40 h of sterilization and starvation, *A. aphelenchus* were washed from plates with K-medium and transferred onto fungi.

Nematode bioassays

***C. elegans* liquid assay.** Liquid assays for active-fraction determination and potency assessment were conducted as previously described (28). In short, *E. coli* was grown in 50 mL LB medium overnight, with orbital shaking at 150 rpm. Cells were pelleted and resuspended in K-medium. OD_{600} was measured and cells diluted to OD_{600} 1.2. An *E. coli* suspension (1.76 mL) was transferred to each well, supplemented with 200 µL nematode suspension and 40 µL test substance resuspended in MeOH. For fraction-testing cured and symbiotic *M. verticillata* NRRL6337 cultures, grown for 16 days on 400 mL PDA, were extracted with ethyl acetate as described above. The extract was fractionated using a preparative HPLC under following conditions: A: H_2O + 0.01 % TFA, B: methanol; 15 % B for 5 min, 15–100 % B in 35 min, 100 % B for 10 min, 15 mL min^{-1} (Phenomenex, Luna, 10 µm, C18(2), 100 Å, 250 × 21.2 mm). Eight fractions were collected (every 5 min one fraction) with an additional fraction at 100 % B for 10 min. The fractions were dried in vacuum and resuspended in 1 mL MeOH. For potency assessment pure **4**, dissolved in MeOH was tested in following concentrations: 0.1 µg mL^{-1}, 0.3 µg mL^{-1}, 1 µg mL^{-1}, 3 µg mL^{-1}, 10 µg mL^{-1}, 30 µg mL^{-1}, 100 µg mL^{-1} and 300 µg mL^{-1}. The nematode suspension was prepared as described above. The OD_{600} of each well was measured at the start of the experiment and compared to the OD_{600} after 4 days of incubation at 20 °C and orbital shaking at 50 rpm. Methanol, 18 mM boric acid (dissolved in H_2O) and K-medium were used as controls for substance activity and a well without nematodes was used as a control for natural degradation-control of the bacteria. All steps were carried out under sterile conditions. For IC_{50} calculation GraphPad Prism 8 was used. The OD_{600} after 4 days of incubation with pure Methanol was used as an infinite small concentration of necroxime, whereas an infinite high concentration was set to 100 % starting OD_{600} *E. coli* (comparable with *E. coli* control).
The results of three biological replicates with each three technical replicates were analyzed for fraction assessment, whereas five biological replicates with three technical replicates each were used for potency determination of necroxime D (**4**).

***A. avenae* co-incubation assay.** A small amount of hyphae of each tested *Mortierella* strain was transferred to a PDA plate and incubated at 24 °C overnight. Nematodes from one plate were sterilized and starved as described above. After one washing step, nematodes were resuspended in 300 µL K-medium and aliquots of 50 µL were distributed onto the fresh fungal cultures. Plates were dried and controlled for living nematodes, before they were incubated for 17–24 days at 20 °C. For the evaluation, nematodes were harvested via Baermann funneling as described above. Funneled *A. avenae* were recovered on 1.5 % agar plates containing 200 mM geneticin and 50 µg mL^{-1} kanamycin overnight and subsequently investigated with a Zeiss Axio Zoom.V16 Stereomicroscope (Zeiss, Oberkochen, Germany) and a magnitude of 25 (Figure S 9). For plates with expected high nematode numbers, only a quarter was transferred onto the agar plates and the actual nematode number recalculated. The number of nematodes on the first frame of each video was counted manually. The mean of the cured *M. verticillata* NRRL 6337 cultures was set to 100 % for each biological replicate and the numbers of the other co-cultivations were calculated in relation to these 100 % (Table S 5). For statistical evaluation and significance evaluation a two-way analysis of variance followed by Tukey's multiple comparisons test with GraphPad Prism 8 was performed (Table S 6). Time series were bioinformatically analyzed as described later. Remaining plates were extracted with ethyl acetate to control the metabolite production. All steps were carried out under sterile conditions. The results of three biological replicates with each three technical replicates were used for analysis.

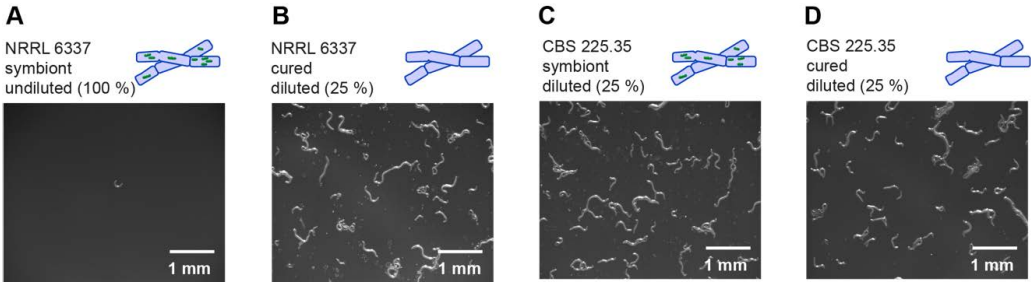


Figure S 9. Exemplary images of harvested nematodes from diverse co-cultivations with a magnification of 25 ×. A) Nematodes grown on symbiotic *M. verticillata* NRRL 6337. Sample of nematodes was undiluted. B) Nematodes of co-culture with a cured *M. verticillata* NRRL 6337. Sample of nematodes represent one quarter of the total nematode count, harvested from the co-incubation. C) Nematodes of co-culture with a symbiotic *M. verticillata* CBS 225.35. Sample of nematodes represent one quarter of the total nematode count, harvested from the co-incubation. D) Nematodes of co-culture with a cured *M. verticillata* CBS 225.35. Sample of nematodes represent one quarter of the total nematode count, harvested from the co-incubation.

Table S 5. Relative numbers of nematodes washed out of *Mortierella* and *A. avenae* co-cultures. Numbers correspond to nematode-counts for each technical replicate from a stereomicroscopic frame with a magnitude of 25. Mean of absolute nematode counts from NRRL 6337 cured were set to 100 % for each biological replicate and relative numbers were calculated based on corresponding 100 %.

Fungal strain	Biological replicate 1 Relative nematode numbers [%]			Biological replicate 2 Relative nematode numbers [%]			Biological replicate 3 Relative nematode numbers [%]		
NRRL 6337 symbiont	17.500	5.000	7.500	0.329	0.658	0.329	1.148	1.913	0.765
NRRL 6337 cured	107.500	120.000	72.500	69.737	152.632	77.632	122.449	113.265	64.286
CBS 225.35 symbiont	155.000	117.500	207.500	128.947	119.737	110.526	52.041	56.633	58.163
CBS 225.35 cured	250.000	187.500	77.500	31.579	26.316	121.053	45.918	73.469	75.000

Table S 6. Results of the statistical analysis (two-way analysis of variance and Turkey's multiple comparison test) representing the significant differences in nematode-counts of the tested co-cultures.

Turkey's multiple comparison test	Mean difference	95 % Confidence interval of difference	Adjusted *p* value	Significance
NRRL 6337 symbiont vs. NRRL 6337 cured	–96.10	–128.1 to –63.5	<0.0001	****
NRRL 6337 symbiont vs. CBS 225.35 symbiont	–107.90	–159.3 to –56.5	0.0007	***
NRRL 6337 symbiont vs. CBS 225.35 cured	–94.80	–170.5 to –19.1	0.0164	*
NRRL 6337 cured vs. CBS 225.35 symbiont	–11.78	–80.5 to 56.9	0.9441	ns
NRRL 6337 cured vs. CBS 225.35 cured	1.30	–84.5 to 87.1	>0.9999	ns
CBS 225.35 symbiont vs. CBS 225.35 cured	13.08	–69.8 to 95.9	0.9555	ns

Image analysis and mathematical modeling of *A. avenae* viability in fungal-nematodal co-incubations. Time series from *A. avenae* co-incubation assay were further analyzed regarding the mobility ratios of the nematodes, which could be harvested from the co-incubation assays. The native Zeiss format CZI files of the nematode samples were analyzed using the 25 × magnification subset of images. This magnification provided a suitable number of observed nematodes per field of view to assure that proper statistical analysis of the kinetic studies could be carried out, whilst maintaining an optical resolution that allowed for precise analysis of the individual nematodes' morphology. The image analysis was carried out using a Fiji (29) macro (ImageJ 1.52s and 1.53c, (30)), modified from our previously established ACAQ-v3 platform (31) to suit the current study. The tracking of the nematodes and further analyses were carried out by our newly developed JIPipe platform (https://www.jipipe.org/). Briefly: the raw CZI images were imported into memory; they were then sharpened, followed by Hessian filtering using the FeatureJ plugin from ImageJ (https://imagescience.org/meijering/software/featurej), where the smallest Hessian eigenvalue images were blurred with a Gaussian filter of 2 pixel radius using a disk element. These images were then thresholded according to the Li algorithm in ImageJ, followed by a closing morphological operator using 2 pixel radius disk elements, as provided by the MorphoLibJ plugin of ImageJ (32). The non-nematode elements of the images were eliminated by applying the Remove Outliers command of ImageJ, first with a radius of 5 pixels, then with 10 pixels; the threshold value was always 50. The clustered objects were separated by applying the watershed algorithm, followed by 2-pixel erosion. The segmented time-series images of nematodes were then loaded into a JIPipe workflow, where the individual nematodes were tracked in time using the "Split into connected components" node. The workflow was then forked; one branch was used to extract the track elements per time point per nematode, whereas the parallel branch summed up the total area covered by each individual nematode during the complete observation time. The ratio between a nematode's area and the total area covered by the same nematode during the entire time series was then used for further analysis. Here, when a nematode was motionless during the entire time series, this ratio would equal one, whereas a fast-moving nematode would be characterized by a high value of this ratio. The faster the nematode's movement, the higher the ratio. The time series of values was further processed to calculate its mean and standard deviation in order to characterize each track in terms of the underlying nematode's health status. We defined that a "dead or paralyzed" nematode would result in a ratio between 1.0 and 1.39 for its track, whereas a "live" nematode would have a ratio above 1.4 (Figure S 10). All results were saved in CSV file formats for further analysis.

When comparing the mobility ratios of the nematodes based on the method described above, the effect sizes using Hedges' g and Cohen's d were calculated for each pair of conditions, using the Effect Size Calculator from Social Science Statistics (www.socscistatistics.com) (Table S 7 and S 8). Typically, effect sizes below 0.2 are considered to indicate a trivial difference, whereas values above 0.8 indicate a real difference between the compared distributions.

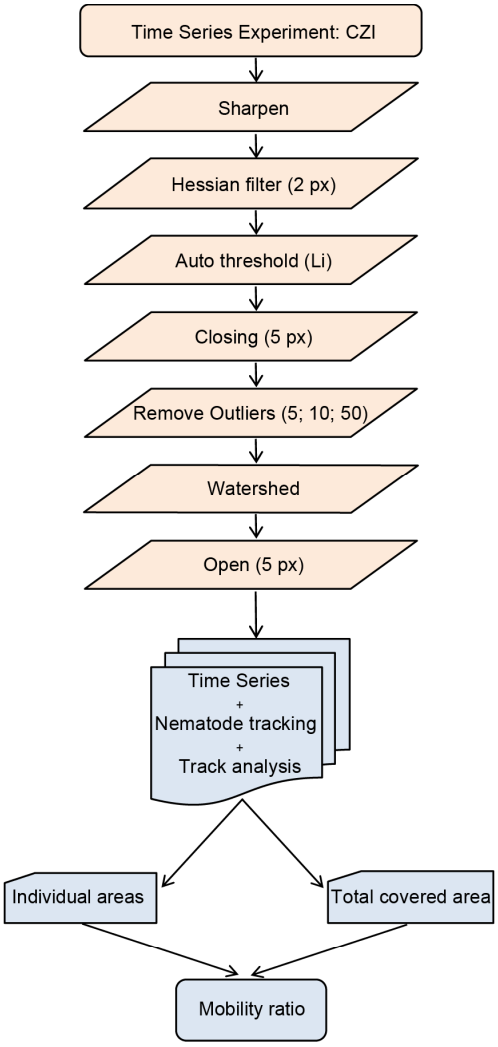


Figure S 10**.** Exemplified steps during image analysis and mathematical modeling of experiments regarding *A. avenae* viability in fungal-nematodal co-incubations.

Table S 7. Overview of analyzed of the different fungal-nematodal co-cultivations with mean numbers in percent of live nematodes and corresponding standard deviation.

Fungal strain	Analyzed	Mean value alive [%]	Standard deviation of mean value [%]
NRRL 6337 symbiont	238	33.7	28.0
NRRL 6337 cured	487	61.9	9.7
CBS 225.35 symbiont	176	63.2	3.2
CBS 225.35 cured	568	73.9	17.2

Table S 8. Effect size calculations for the mobility ratios of the different fungal-nematodal co-cultivations.

Compared populations	Hedges' *g*	Cohen's *d*	Strength of effect
NRRL 6337 symbiont vs. NRRL 6337 cured	1.58	1.35	Very large
NRRL 6337 symbiont vs. CBS 225.35 symbiont	1.38	1.48	Very large
NRRL 6337 symbiont vs. CBS 225.35 cured	1.92	1.73	Very large
NRRL 6337 cured vs. CBS 225.35 symbiont	0.15	0.18	Very small
NRRL 6337 cured vs. CBS 225.35 cured	0.84	0.86	Moderate
CBS 225.35 symbiont vs. CBS 225.35 cured	0.71	0.86	Moderate

***A. avenae* chemical complementation assay.** A small amount of hyphae of cured or symbiotic *M. verticillata* NRRL 6337 strains was transferred onto 1 mL PDA filled into 12-well plates and incubated overnight at 26 °C. On cultures inoculated for chemical complementation 11.4 µg (25 µM), 22.8 µg (50 µM), 50 µg (109 µM) or 100 µg (219 µM) necroxime D (**4**) dissolved in 200 µL 50 % MeOH were applied and dried under sterile conditions. Control cultures were overlaid with 200 µL 50 % MeOH and dried. Aliquots of 50 µL nematode suspension, prepared as described above, were distributed onto fungi, dried and co-incubated for 14 days at 20 °C. For evaluation, co-culture was removed from the wells and washed in 5 mL K-medium overnight. Medium including the washed nematodes was filtered through miracloth (Merck) to avoid agar carry-over and left at 4 °C for 1 h to let nematode settle. Supernatant was discarded and remaining nematodes transferred onto 6-well plates containing 5 mL 1.5 % agar plates with 200 mM geneticin and 50 µg mL^{-1} kanamycin. After plates were dried under sterile conditions, the amount of harvested nematodes from each plate was assessed with a Zeiss Axio Zoom V16 Stereomicroscope (Zeiss, Oberkochen, Germany) (three biological replicates with each three technical replicates) (Figure S 11 and S 12).

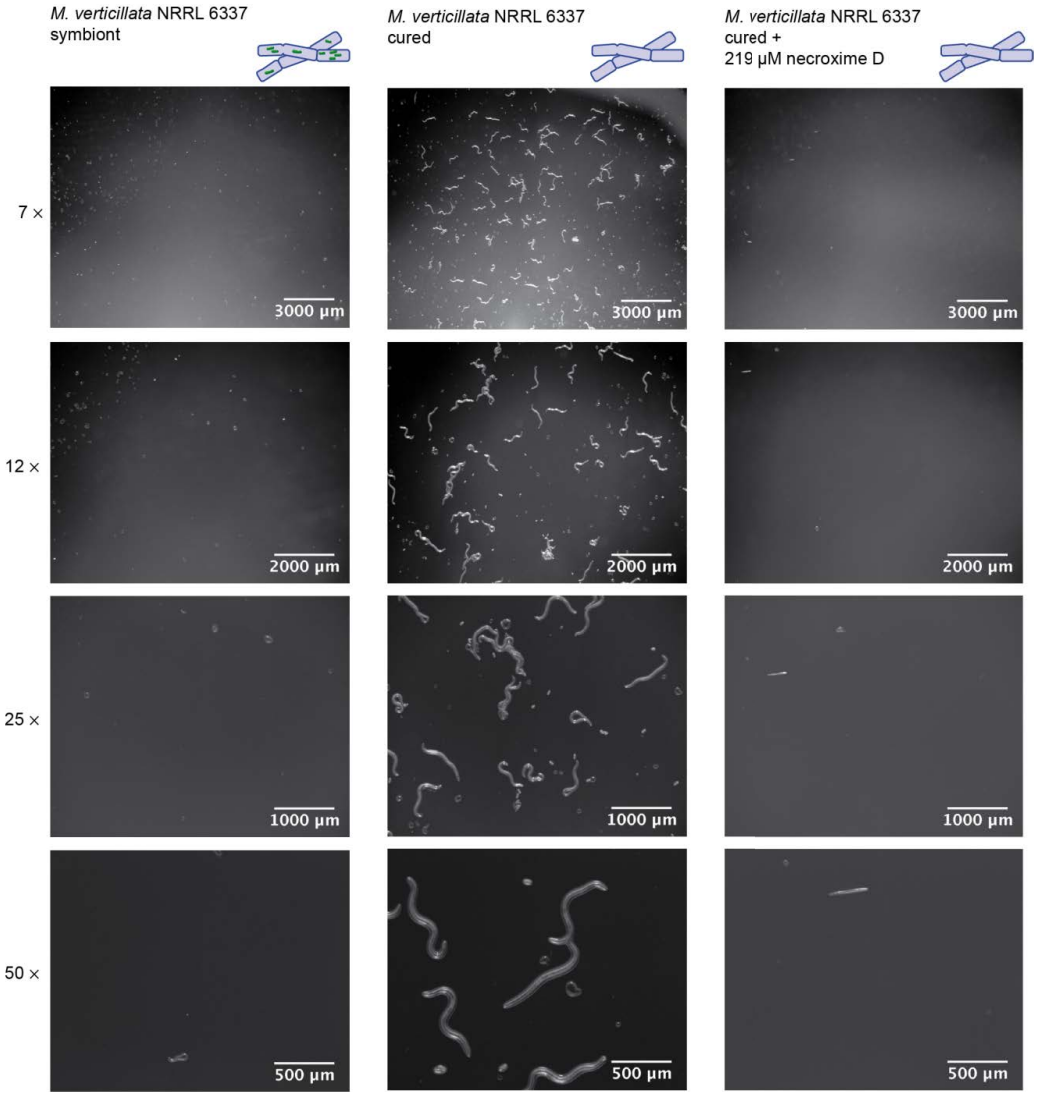


Figure S 11. Stereomicroscopy images in various magnifications of *A. avenae* harvested from symbiotic *M. verticillata* NRRL 6337 cultures, cured *M. verticillata* NRRL 6337 cultures and cured *M. verticillata* NRRL 6337 cultures complemented with 219 µM necroxime D (**4**).

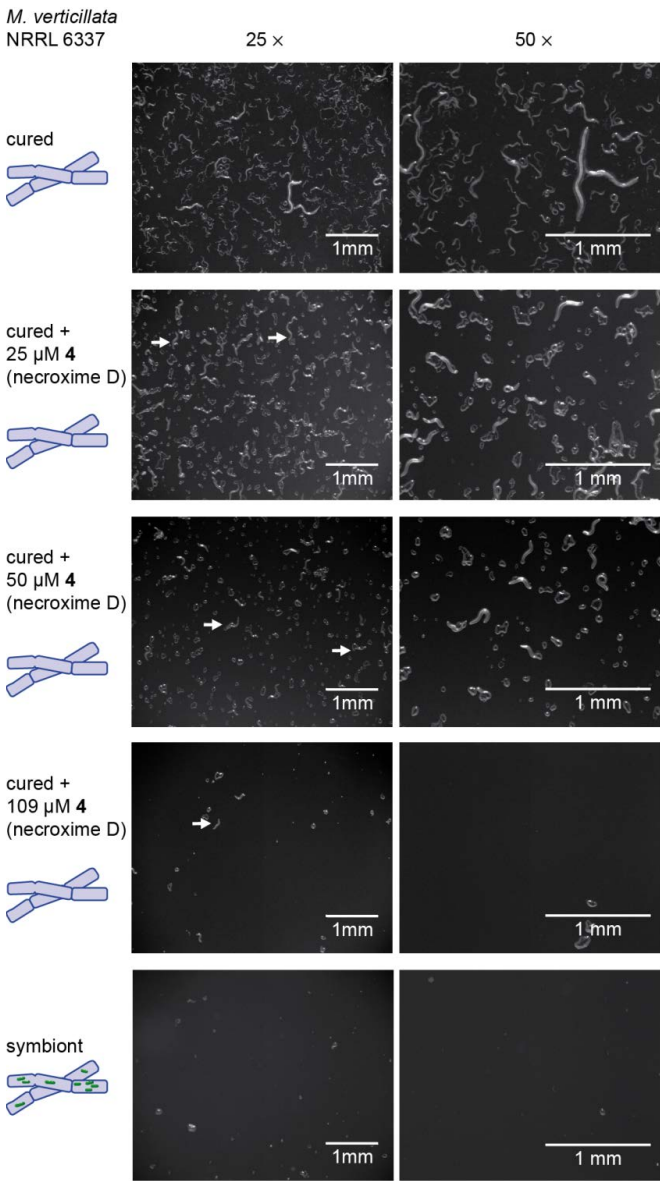


Figure S 12. Stereomicroscopy images in two magnifications of *A. avenae* harvested from symbiotic *M. verticillata* NRRL 6337 cultures, cured *M. verticillata* NRRL 6337 cultures and cured *M. verticillata* NRRL 6337 cultures complemented with 25 µM, 50 µM and 109 µM necroxime D (**4**). White arrows indicate nematodes beside same-sized agar pieces.

MS/MS and NMR data

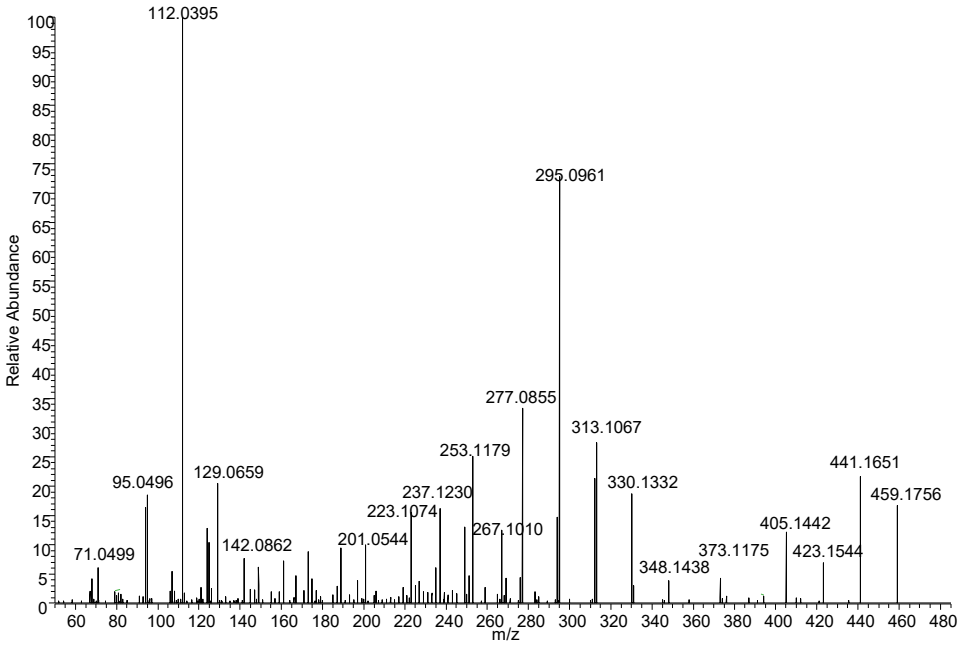


Figure S 13. MS/MS fragmentation pattern of *m/z* 459.1756 $[M+H]^+$ (necroxime C, **3**) in *Burkholderia* sp. strain B8 cultures.

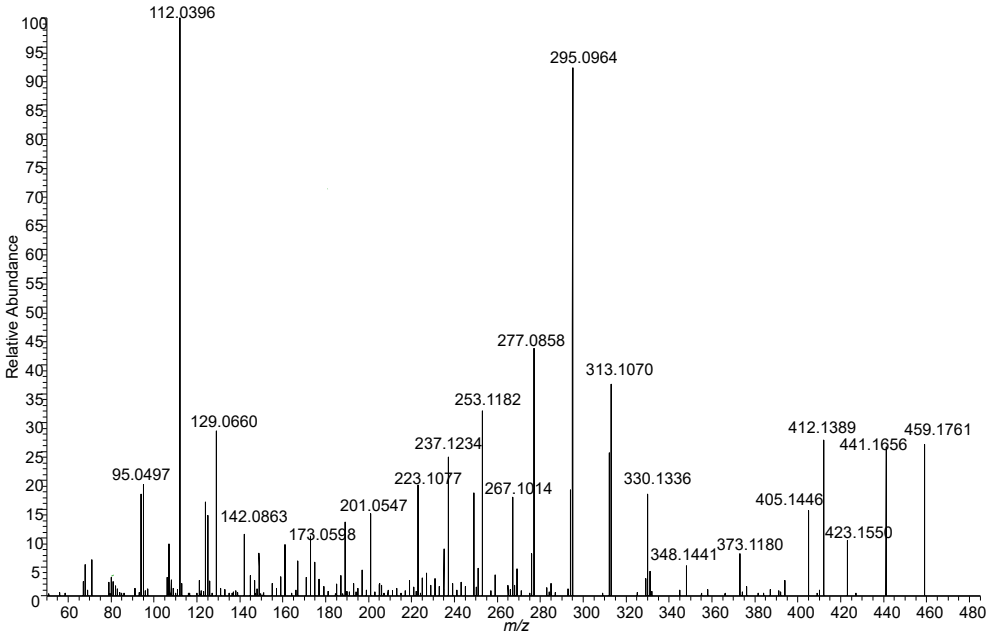


Figure S 14. MS/MS fragmentation pattern of *m/z* 459.1756 $[M+H]^+$ (CJ-13,357, **3**) in *M. verticillata* NRRL 6337 cultures.

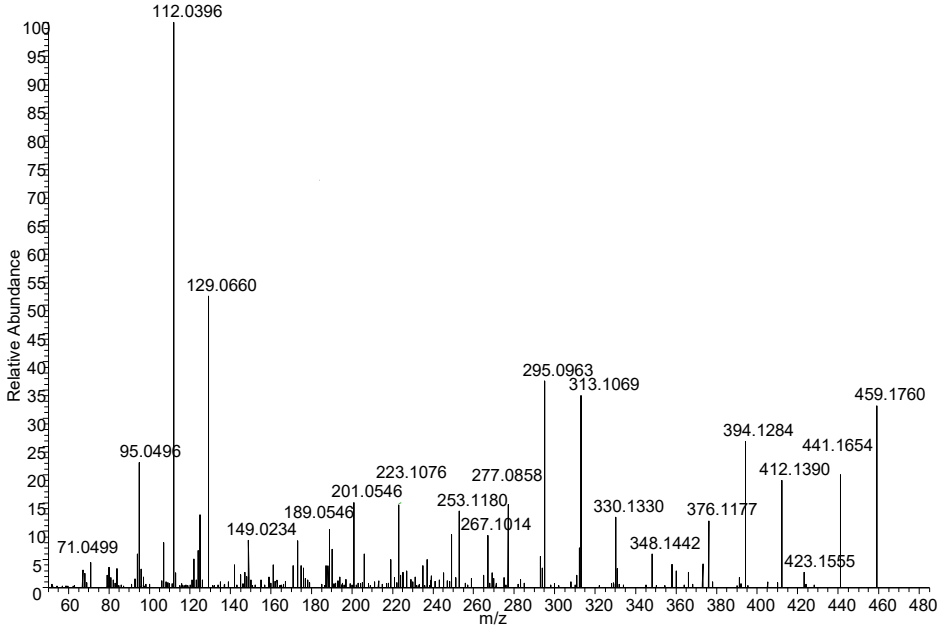


Figure S 15. MS/MS fragmentation pattern of *m/z* 459.1760 $[M+H]^+$ (necroxime D, **4**) in *Burkholderia* sp. strain B8 cultures.

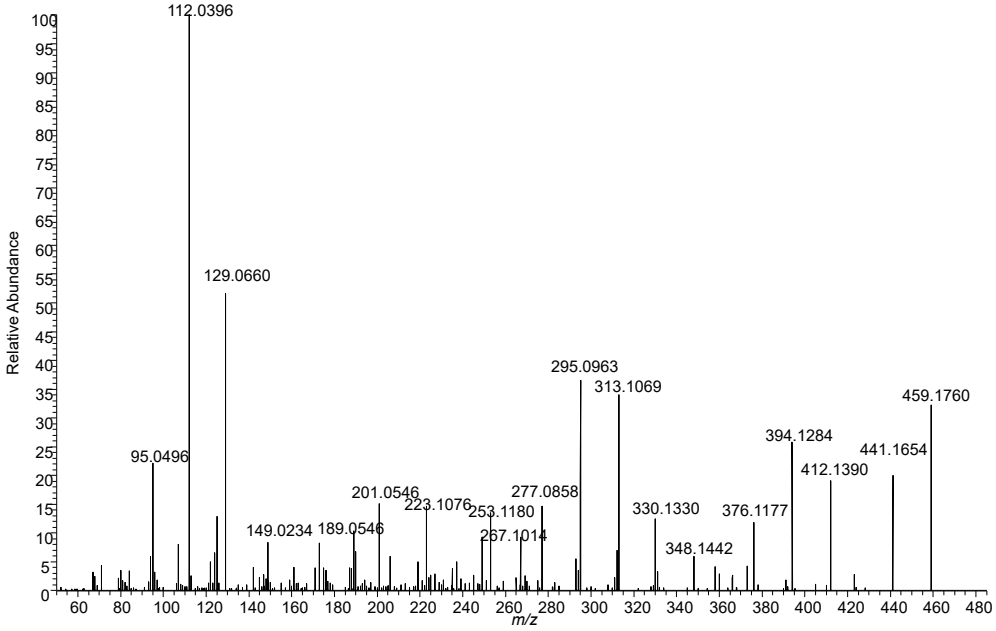


Figure S 16. MS/MS fragmentation pattern of *m/z* 459.1756 $[M+H]^+$ (CJ-12,950, **4**) in *M. verticillata* NRRL 6337 cultures.

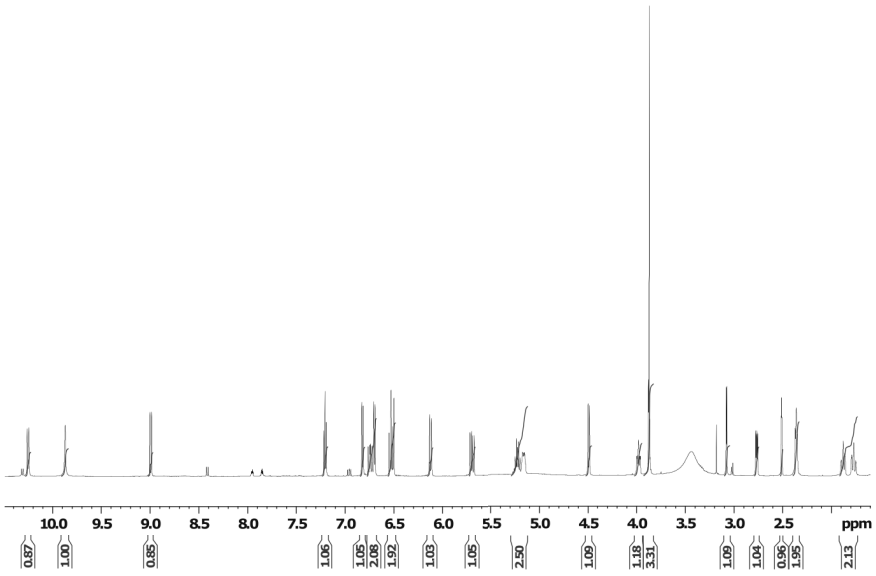


Figure S 17. [1]H-NMR of necroxime D (**4**) isolated from *M. verticillata* NRRL 6337 cultures.

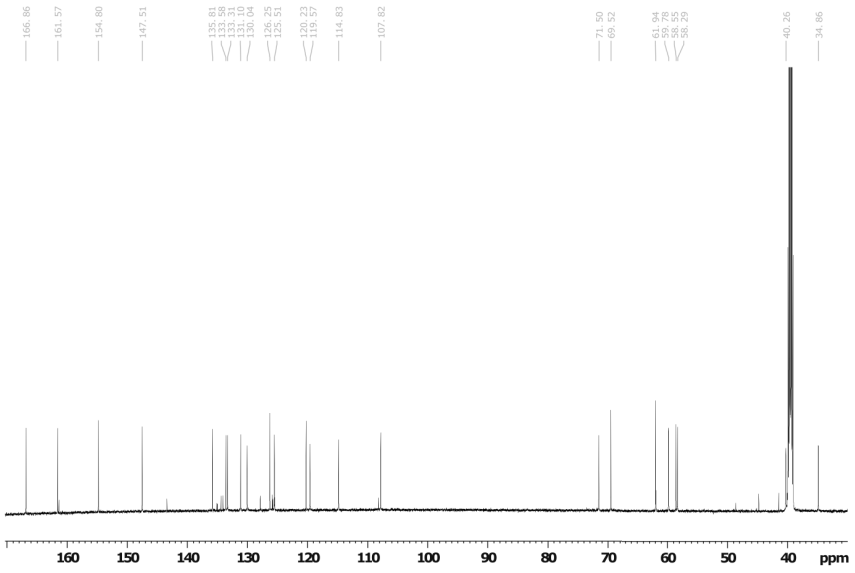


Figure S 18. [13]C-NMR of necroxime D (**4**) isolated from *M. verticillata* NRRL 6337 cultures.

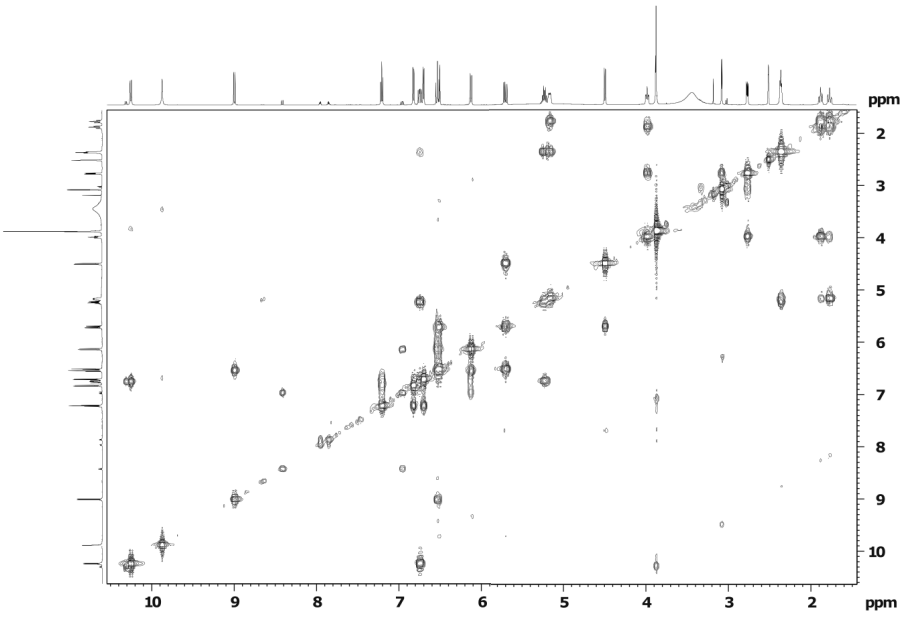


Figure S 19. ^{1}H-^{1}H-COSY-NMR of necroxime D (**4**) isolated from *M. verticillata* NRRL 6337 cultures.

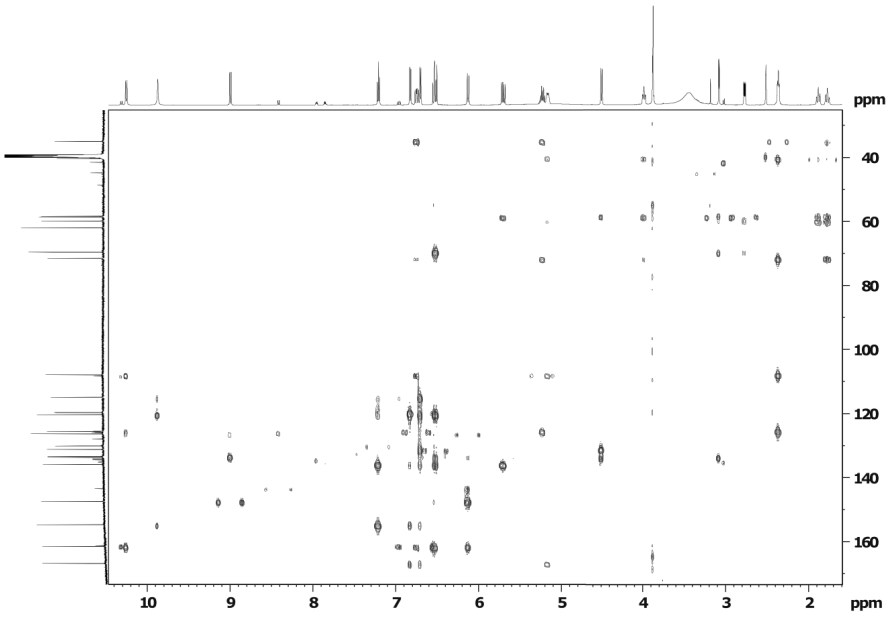


Figure S 20. ^{1}H-^{13}C-HMBC-NMR of necroxime D (**4**) isolated from *M. verticillata* NRRL 6337 cultures.

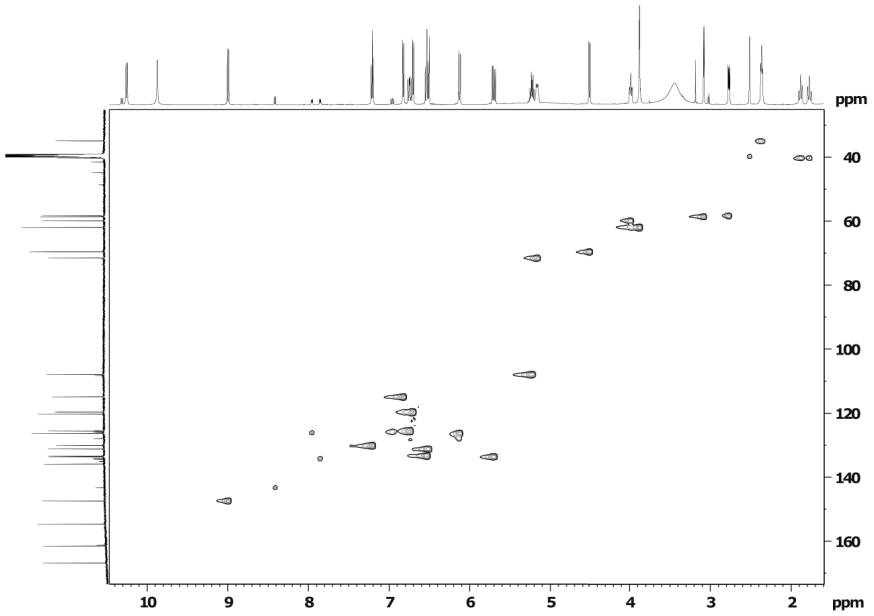


Figure S 21**.** ^{1}H-^{13}C-HSQC-NMR of necroxime D (**4**) isolated from *M. verticillata* NRRL 6337 cultures.

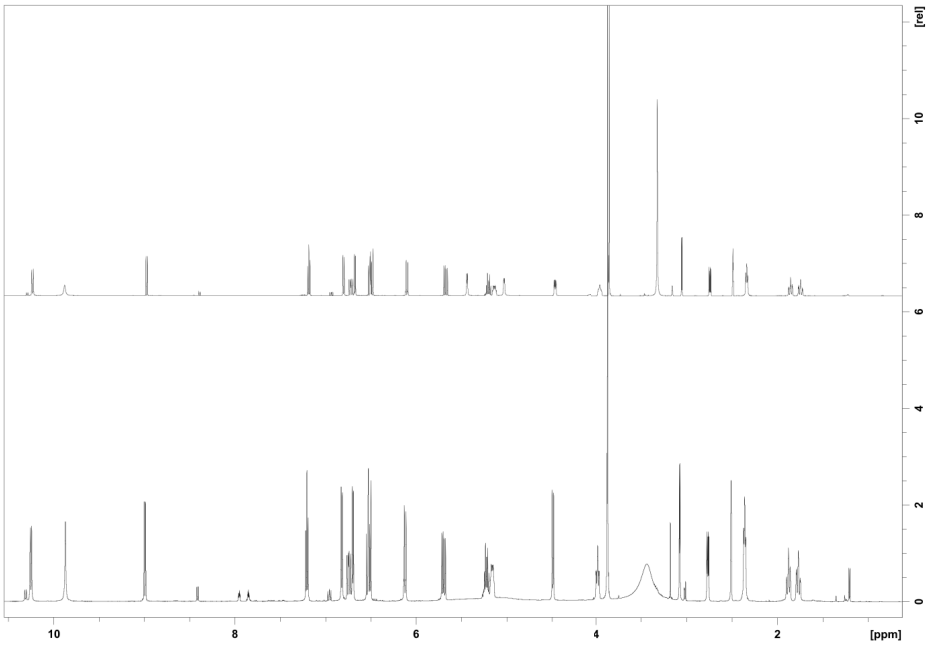


Figure S 22. Comparison of ^{1}H-NMR of necroxime D (**4**) isolated from *M. verticillata* NRRL 6337 cultures and ^{1}H-NMR of necroxime D (**4**) isolated from *Burkholderia* sp. strain B8 cultures.

Table S 9. Comparison of the chemical shifts from the NMR data of necroxime D (CJ-12,950, **4**) isolated from *M. verticillata* NRRL 6337 cultures and NMR data of necroxime C and D (**3** and **4**) isolated from *Burkholderia* sp. strain B8 cultures. Main differences between *E*/*Z* isomers are marked in red.

	Necroxime C (3)		**Necroxime D (4)**		**CJ-12,950 (4)**	
Pos.	δ_C [ppm]	δ_H [ppm]; Signal (*J* [Hz])	δ_C [ppm]	δ_H [ppm]; Signal (*J* [Hz])	δ_C [ppm]	δ_H [ppm]; Signal (*J* [Hz])
1	167.0	-	166.8	-	166.9	-
2	120.3	-	120.2	-	120.2	-
3	154.9	-	154.8	-	154.7	-
4	119.5	6.81; 1 H d (*8.0*)	114.8	6.80; 1 H d (*8.2*)	114.9	6.82; 1 H d (*8.6*)
5	130.0	7.20; 1 H t (*7.8*)	130.0	7.18; 1 H t (*8.0*)	130.0	7.21; 1 H t (*7.9*)
6	120.2	6.69; 1 H d (*7.4*)	119.5	6.68; 1 H d (*7.5*)	119.6	6.69; 1 H d (*7.6*)
7	135.8	-	135.8	-	135.8	-
8	131.1	6.51; 1 H d (*16.5*)	131.1	6.49; 1 H d (*16.2*)	131.1	6.51; 1 H d (*16.6*)
9	133.7	5.69; 1 H dd (*9.2; 16.4*)	133.6	5.67; 1 H dd (*9.2; 16.9*)	133.8	5.70; 1 H dd (*9.2; 16.2*)
10	69.5	4.49; 1 H dd (*4.2; 9.2*)	69.5	4.47; 1 H dd (*4.7; 9.1*)	69.5	4.49; 1 H d (*9.0*)
11	58.6	3.07; 1 H d (*4.3*)	58.5	3.05; 1 H d (*4.3*)	58.5	3.07; 1 H d (*4.2*)
12	58.3	2.76; 1 H dd (*4.2; 8.7*)	58.3	2.74; 1 H dd (*4.2; 8.5*)	58.2	2.77; 1 H dd (*4.1; 8.7*)
13	59.9	3.97; 1 H m	59.8	3.96; 1 H m	59.8	3.98; 1 H m
14	40.3	1.87; 1 H m	40.3	1.86; 1 H m	40.3	1.88; 1 H m
		1.77; 1 H m		1.74; 1 H m		1.76; 1 H m
15	71.5	5.15; 1 H m	71.5	5.14; 1 H m	71.5	5.16; 1 H m
16	34.7	2.35; 2 H t (*6.8*)	34.9	2.34; 2 H t (*7.1*)	34.9	2.36; 2 H t (*6.7*)
17	107.3	5.24; 1 H dt (*8.0; 14.5*)	107.8	5.20; 1 H dt (*7.0; 14.2*)	107.9	5.22; 1 H dt (*14.2; 7.2*)
18	125.5	6.76; 1 H dd (*10.3; 14.5*)	125.5	6.72; 1 H dd (*10.4; 14.4*)	125.5	6.75; 1 H dd (*10.2; 14.5*)
NH	-	10.23; 1 H d (*10.2*)	-	10.23; 1 H d (*10.3*)	-	10.25 1 H d (*10.1*)
19	161.3	-	161.6	-	161.6	-
20	130.5	6.39; 1 H d (*15.3*)	126.2	6.10; 1 H d (*11.6*)	126.2	6.12; 1 H d (*11.5*)
21	132.8	7.00; 1 H dd (*10.2; 15.5*)	133.3	6.52; 1 H dd (*10.2; 11.3*)	133.3	6.53; 1 H m
22	148.9	8.05; 1 H d (*10.6*)	147.5	8.97; 1 H d (*10.5*)	147.5	8.99 1 H d (*10.3*)
23	62.0	3.88; 3 H s	61.9	3.85; 3 H s	61.9	3.87; 3 H s
3-OH	-	9.86; 1 H br	-	9.88; 1 H br		9.87 1 H br
10-OH	-	5.46; 1 H br	-	5.44; 1 H d (*5.3*)		-

Legends for Movies

Movie S 1. *A. avenae* harvested from a co-culture with symbiotic *M. verticillata* NRRL 6337.

Movie S 2. *A. avenae* harvested from a co-culture with cured *M. verticillata* NRRL 6337.

SI References

1. P. Estrada-de los Santos *et al.*, Whole genome analyses suggests that *Burkholderia* sensu lato contains two additional novel genera (*Mycetohabitans* gen. nov., and *Trinickia* gen. nov.): implications for the evolution of diazotrophy and nodulation in the Burkholderiaceae. *Genes* **9**, 389 (2018).
2. F. Madeira *et al.*, The EMBL-EBI search and sequence analysis tools APIs in 2019. *Nucleic Acids Res.* **47**, W636-W641 (2019).
3. J. Trifinopoulos, L. T. Nguyen, A. von Haeseler, B. Q. Minh, W-IQ-TREE: a fast online phylogenetic tool for maximum likelihood analysis. *Nucleic Acids Res.* **44**, W232-235 (2016).
4. Y. Takashima *et al.*, Prevalence and intra-family phylogenetic divergence of *Burkholderiaceae*-related endobacteria associated with species of *Mortierella*. *Microbes Environ.* **33**, 417-427 (2018).
5. S. Ohshima *et al.*, *Mycoavidus cysteinexigens* gen. nov., sp. nov., an endohyphal bacterium isolated from a soil isolate of the fungus *Mortierella elongata*. *Int. J. Syst. Evol. Microbiol.* **66**, 2052-2057 (2016).
6. R. R. Wick, L. M. Judd, C. L. Gorrie, K. E. Holt, Unicycler: resolving bacterial genome assemblies from short and long sequencing reads. *PLoS Comput. Bio.* **13**, e1005595 (2017).
7. R. R. Wick, L. M. Judd, C. L. Gorrie, K. E. Holt, Completing bacterial genome assemblies with multiplex MinION sequencing. *Microb. Genom.* **3**, e000132 (2017).
8. K. Blin *et al.*, antiSMASH 5.0: updates to the secondary metabolite genome mining pipeline. *Nucleic Acids Res.* (2019).
9. G. Bashiri, A. M. Rehan, D. R. Greenwood, J. M. Dickson, E. N. Baker, Metabolic engineering of cofactor F420 production in *Mycobacterium smegmatis*. *PLoS One* **5**, e15803 (2010).
10. D. Braga *et al.*, Metabolic pathway rerouting in *Paraburkholderia rhizoxinica* evolved long-overlooked derivatives of coenzyme F420. *ACS Chem. Biol.* **14**, 2088-2094 (2019).
11. L. D. Eirich, G. D. Vogels, R. S. Wolfe, Proposed structure for coenzyme F420 from *Methanobacterium*. *Biochemistry* **17**, 4583-4593 (1978).
12. B. O. Bachmann, J. Ravel, Methods for *in silico* prediction of microbial polyketide and nonribosomal peptide biosynthetic pathways from DNA sequence data. *Methods Enzymol.* **458**, 181-217 (2009).
13. J. Uehling *et al.*, Comparative genomics of *Mortierella elongata* and its bacterial endosymbiont *Mycoavidus cysteinexigens*. *Environ. Microbiol.* **19**, 2964-2983 (2017).
14. Y. Guo *et al.*, *Mycoavidus* sp. Strain B2-EB: comparative genomics reveals minimal genomic features required by a cultivable Burkholderiaceae-related endofungal bacterium. *Appl. Environ. Microbiol.* **86**, e01018-01020 (2020).
15. D. Sharmin *et al.*, Comparative genomic insights into endofungal lifestyles of two bacterial endosymbionts, *Mycoavidus cysteinexigens* and *Burkholderia rhizoxinica*. *Microbes Environ.*, ME17138 (2018).
16. T. Seemann, Prokka: rapid prokaryotic genome annotation. *Bioinformatics* **30**, 2068-2069 (2014).
17. A. J. Page *et al.*, Roary: rapid large-scale prokaryote pan genome analysis. *Bioinformatics* **31**, 3691-3693 (2015).
18. M. Krzywinski *et al.*, Circos: an information aesthetic for comparative genomics. *Genome Res.* **19**, 1639-1645 (2009).
19. J. Uehling *et al.*, Comparative genomics of *Mortierella elongata* and its bacterial endosymbiont *Mycoavidus cysteinexigens*. *Environ. Microbiol.* **19**, 2964-2983 (2017).
20. M. N. Price, A. M. Deutschbauer, A. P. Arkin, GapMind: Automated Annotation of Amino Acid Biosynthesis. *mSystems* **5**, e00291-00220 (2020).
21. K. A. Dekker *et al.*, Novel lactone compounds from *Mortierella verticillata* that induce the human low density lipoprotein receptor gene: Fermentation, isolation, structural elucidation and biological activities. *J. Antibiot.* **51**, 14-20 (1998).

22. S. P. Niehs *et al.*, Mining symbionts of a spider-transmitted fungus illuminates uncharted biosynthetic pathways to cytotoxic benzolactones. *Angew. Chem. Int. Ed.* **59**, 7766-7771 (2020).
23. S. Kumar, G. Stecher, K. Tamura, MEGA7: Molecular Evolutionary Genetics Analysis Version 7.0 for Bigger Datasets. *Mol. Biol. Evol.* **33**, 1870-1874 (2016).
24. J. D. Thompson, D. G. Higgins, T. J. Gibson, CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.* **22**, 4673-4680 (1994).
25. P. Caffrey, Conserved amino acid residues correlating with ketoreductase stereospecificity in modular polyketide synthases. *ChemBioChem* **4**, 654-657 (2003).
26. J. H. Sulston, J., The nematode *Caenorhabditis elegans*. *Cold Spring Harbor Laboratory* **17**, 988 (1988).
27. S. Bleuler-Martínez *et al.*, A lectin-mediated resistance of higher fungi against predators and parasites. *Mol. Ecol.* **20**, 3056-3070 (2011).
28. M. P. Smith *et al.*, A liquid-based method for the assessment of bacterial pathogenicity using the nematode *Caenorhabditis elegans*. *FEMS Microbiol. Lett.* **210**, 181-185 (2002).
29. J. Schindelin *et al.*, Fiji: an open-source platform for biological-image analysis. *Nat. Methods* **9**, 676-682 (2012).
30. C. T. Rueden *et al.*, ImageJ2: ImageJ for the next generation of scientific image data. *BMC Bioinform.* **18**, 529 (2017).
31. Z. Cseresnyes, K. Kraibooj, M. T. Figge, Hessian-based quantitative image analysis of host-pathogen confrontation assays. *Cytometry A* **93**, 346-356 (2018).
32. D. Legland, I. Arganda-Carreras, P. Andrey, MorphoLibJ: integrated library and plugins for mathematical morphology with Image J. *Bioinformatics* **32**, 3532-3534 (2016).

4.4 MISA++: A standardized interface for automated bioimage analysis

Manuskript Nr. 4

Titel des Manuskriptes: MISA++: A standardized interface for automated bioimage analysis

Autoren: Ruman Gerst, Anna Medyukhina, Marc Thilo Figge

Bibliographische Informationen: Gerst, R., Medyukhina, A., & Figge, M. T. (2020). MISA++: A standardized interface for automated bioimage analysis. SoftwareX, 11, 100405.

Der Kandidat / Die Kandidatin ist (bitte ankreuzen)

☒ Erstautor/-in, ☐ Ko-Erstautor/-in, ☐ Korresp. Autor/-in, ☐ Koautor/-in.

Status: Veröffentlicht in SoftwareX

Anteile (in %) der Autoren / der Autorinnen an den vorgegebenen Kategorien der Publikation

Autor/-in	Konzeptionell	Datenanalyse	Experimentell	Verfassen des Manuskriptes	Bereitstellung von Material
Ruman Gerst	70 %	70 %	70 %	35 %	0 %
Anna Medyukhina	15 %	15 %	15 %	35 %	0 %
Marc Thilo Figge	15 %	15 %	15 %	30 %	100 %
Summe:	100 %	100 %	100 %	100 %	100 %

Unterschrift Kandidat/-in

Unterschrift Betreuer/-in (Mitglied der Fakultät)

SoftwareX 11 (2020) 100405

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

MISA++: A standardized interface for automated bioimage analysis

Ruman Gerst [a,b], Anna Medyukhina [a], Marc Thilo Figge [a,c,*]

[a] Applied Systems Biology, Leibniz Institute for Natural Product Research and Infection Biology – Hans-Knöll-Institute, Jena, Germany
[b] Faculty of Biological Sciences, Friedrich-Schiller-University Jena, Germany
[c] Institute of Microbiology, Faculty of Biological Sciences, Friedrich-Schiller-University Jena, Germany

ARTICLE INFO

Article history:
Received 4 September 2019
Received in revised form 16 January 2020
Accepted 19 January 2020

Keywords:
Image processing
Parallelization
Application integration
Light-sheet fluorescence microscopy
Big volume image data

ABSTRACT

Modern imaging techniques, such as lightsheet fluorescence microscopy (LSFM), allow the capture of whole organs in three spatial dimensions. The analysis of these big volume image data requires a combination of user-friendly and highly efficient tools. We here present *MISA++*, an image analysis framework that allows easy integration of custom high-performance *C++* tools into third-party applications via standardized components for parallelization, data and parameter handling, command line interface, and communication with third-party applications. We demonstrate its capabilities by implementing a plugin for *ImageJ* that provides a graphical user interface for any application built with our framework, and a high-performance re-implementation of our *Python*-based algorithm to segment glomeruli in LSFM images of whole murine kidneys.

Code metadata

Current code version	1.0.0.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2019_276
Code Ocean compute capsule	
Legal Code License	BSD-3-Clause
Code versioning system used	Git
Software code languages, tools, and services used	C++, OpenMP, CMake, Java, Python, ImageJ
Compilation requirements, operating environments & dependencies	C++ framework: CMake, Boost, SQLite, OME files, OpenCV, JSON for Modern C++; ImageJ plugin: Maven; Java implementations: Maven, imglib2, DeconvolutionLab2, ImageJ; Python implementations: Snakemake, NumPy, Scikit-image, Mahotas
If available Link to developer documentation/manual	https://applied-systems-biology.github.io/misa-framework/
Support email for questions	thilo.figge@leibniz-hki.de

Software metadata

Current software version	1.0.0.1
Permanent link to executables of this version	https://github.com/applied-systems-biology/misa-framework/releases/download/1.0.0/misaxx-1.0.0.1-sources-bin-aio.zip
Legal Software License	BSD-3-Clause
Computing platforms/Operating Systems	Windows, Linux
Installation requirements & dependencies	
If available, link to user manual	https://applied-systems-biology.github.io/misa-framework/
Support email for questions	thilo.figge@leibniz-hki.de

1. Motivation and significance

Modern imaging techniques, such as light sheet fluorescence microscopy (LSFM), enable quantitative three-dimensional analysis of targeted structures in whole organs. A typical LSFM experiment produces hundreds of gigabytes of 3D image data, making

* Corresponding author at: Applied Systems Biology, Leibniz Institute for Natural Product Research and Infection Biology – Hans-Knöll-Institute, Jena, Germany.
E-mail address: thilo.figge@leibniz-hki.de (M.T. Figge).

https://doi.org/10.1016/j.softx.2020.100405

a manual analysis virtually impossible. The analysis of these big data volumes requires automated and efficient tools that utilize multiple processing cores while managing limited system memory. Such highly efficient tools can be developed with programming languages such as *C++*, *C* or *Julia*. Each language has its own pros and cons: *Julia* supports modern high-level concepts such as functional programming or modularization, but lacks a large software ecosystem of third-party libraries. The opposite is true for *C*: It features a wide range of existing third-party libraries, but lacks modern programming concepts. *C++* fulfills both requirements. It features a large set of third-party software such as *OpenCV* [1], *Insight Toolkit* (*ITK*) [2], *Halide* [3], and – due to a compatibility feature – any library developed in *C*. Additionally, it includes machine-oriented memory management, and high-level concepts like object orientation, functional programming, and modularization. However, development of novel algorithms and pipelines in *C++* is difficult due to a complex syntax, the need of compilation, limited debugging capabilities, and the static program flow. For example, there is no easy way to change algorithm parameters without re-calculating all previous steps.

Tools like *ImageJ* [4], *Icy* [5], *Knime* [6], *Ilastic* [7], *CellProfiler* [8], *Python* (https://python.org/), and *Microscopy Image Browser* (*MIB*) [9] allow easy development of novel algorithms and pipelines by providing a graphical user interface or a simple programming language syntax. Such features on the other hand can make high-performance analysis impractical: Simple scripting languages are not compiled into machine code and can therefore only be optimized to a limited degree. This makes implementations of custom algorithms extremely slow. Graphical user interfaces are not supported by most high-performance servers and come with additional overhead to render the interface.

Analysis of large-volume data often requires the development of novel algorithms and pipelines. This can often be more efficiently done on a small subset of data via a user-friendly tool that, however, may be less optimized. To obtain the results for the full data set within reasonable computation time or by an algorithm adapted for real-time processing, a high-performance re-implementation becomes essential. There are already existing tools like *CLIJ* [10] or *Cython* (http://cython.org) that allow such an optimized implementation. Yet, novel algorithms might require highly customized implementations and usage of high-performance third-party libraries. Features like these are only available in machine-oriented languages such as *C++*.

The disadvantage of such custom high-performance *C++* tools is that they cannot be easily integrated into third-party applications, which not only allow easy statistical analysis of the algorithm results, but often provide proprietary methods for data preprocessing. Today, there are no standards on how to communicate input data, output data, and algorithm parameters. Without such standards, additional development time would be needed for creating a custom-built integration. This causes doubling of similar codes across multiple projects due to incompatibilities and additional effort in documentation and maintenance of the software.

To address the challenge of combining the efficiency of *C++* with user-friendly tools, we developed a framework termed *Modular Image Stack Analysis for C++* (*MISA++*): A platform-independent open source framework that provides a standard for developing efficient, highly parallelized, and modular image analysis tools using established libraries. It allows to create custom software tools with standardized data and parameter handling, parallelization, command line interface, and documentation. Any tool developed with *MISA++* can be easily integrated into user-friendly software.

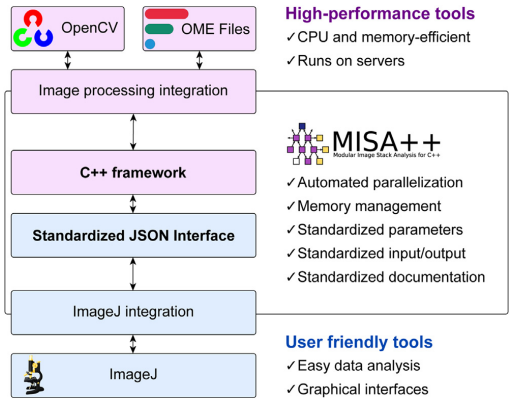


Fig. 1. *MISA++* consists of two core components: A *C++* framework and a standardized *JSON* interface. The high-performance *C++* framework allows integration of third-party libraries like *OpenCV* or *OME Files*. The interface allows easy integration of applications developed with the *C++* framework by standardizing parameters, data handling, and documentation.

2. Software architecture

MISA++ consists of two parts: (i) A *C++* framework for developing high-performance image analysis tools – *MISA++* applications – with established libraries like *OpenCV*, and (ii) a standardized interface to communicate between any *MISA++* application and any other software like *ImageJ* (see Fig. 1).

2.1. C++ framework

Our *C++* framework provides ready-to-use components for parallelization, data organization and integration of third-party data types, memory management, parameter handling, creating a command line interface, and interacting with other applications and pipelines. *MISA++* is written in *C++* version 17 and depends on various libraries developed by third-party contributors (see Table S1).

2.1.1. Parallelization

The framework organizes workloads in a directed acyclic graph (DAG) where nodes represent processing steps and edges represent dependencies. Each node contains a workload function and tracks if it already has been executed. A node can only execute its workload if no dependency has an unfinished workload. The framework then iteratively executes unfinished workloads in parallel until all nodes are finished.

MISA++ differentiates between two types of nodes: Tasks and dispatchers. Tasks are workers that contain any atomic and parallelizable workload. Dispatchers are not parallelizable and responsible for creating the DAG structure. To make it possible for developers to easily split the DAG into individual, re-usable applications and software-libraries, the DAG is constructed iteratively during the runtime: Dispatcher nodes only instantiate its direct dependencies — either tasks or other dispatchers. An example of such a graph construction is presented in Fig. S1.

2.1.2. Data management

Input and output data is organized in data structures termed caches. They fulfill four roles (see Fig. S2): (i) Integration of third-party data types into *MISA++*, (ii) automated memory-

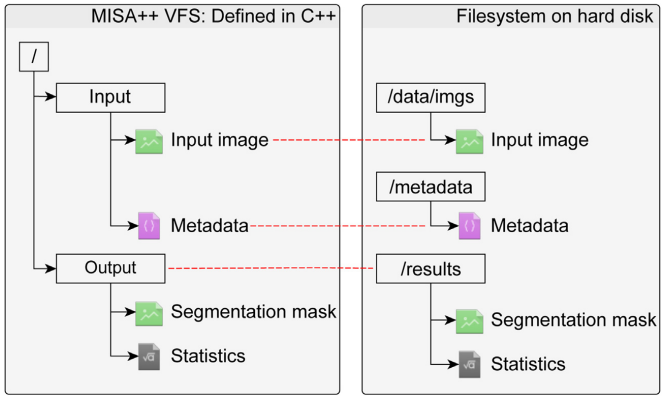


Fig. 2. *MISA++* applications organize files in a virtual file system (VFS). Its structure is defined in *C++* code. Entries in the VFS can be linked to any entry in the hard disk file system (red dashed lines).

management by storing unused data on hard drive, (iii) thread-safe data reading, writing and modification, and (iv) managing quantification results.

Caches encapsulate the process of reading and writing data from an automatically assigned location on the hard disk. Hard drive locations are managed in a virtual file system (VFS; see Fig. 2). The framework then maps VFS paths to locations on the hard drive, based on information provided via the application parameters. This makes it easy to adapt a *MISA++* application to different computers and third-party programs. Another benefit is that the framework can automatically de-allocate unused data from memory and store it on hard disk to lower system requirements.

In a multi-threaded environment, it is essential to prevent behavior like parallel writing into the same memory location. Breaking this thread-safety would cause wrong calculation results or crashes. Caches handle data access from multiple threads by blocking data writing until all reading threads have finished their work.

Caches allow attachment of custom metadata objects for thread-safe intermediate results storage, or to export quantification data. All attached results are automatically stored in a standardized format (see Section 2.2.2).

2.1.3. Parameter handling

All parameters are provided via a parameter file in *JSON* format (http://json.org/) and accessible via thread-safe functions. These methods also automatically assign a location in the *JSON* file and de-serialize *JSON* data, allow developers to document parameters, and set constraints. There are three types of parameters that can be defined in code: (i) DAG-node specific parameters, termed algorithm parameters, (ii) sample-specific parameters, and (iii) global parameters. Global parameters are used for general settings like the number of threads, while sample parameters contain sample metadata. Algorithm parameters are specific to one DAG node and allow users to modify the behavior of workloads.

2.1.4. Command line interface

MISA++ comes with components that allow encapsulation of any DAG root into a ready-to-use command line interface (CLI). The CLI handles tasks like reading the parameter file, building the VFS, and executing DAG node workloads. Additionally, it is able to query information provided in *C++* code to generate a documentation of the *MISA++* application (see Section 2.2.3).

2.2. Standardized JSON interface

The components described in the previous section are standardized, so that developers, users, and other software can easily interact with *MISA++* applications. Components of this interface are the standardized CLI (see Section 2.1.4), the standardized VFS (see Section 2.1.2), and multiple standards on how to communicate information like metadata or parameters via the *JSON* format. The full interface documentation can be accessed online at https://applied-systems-biology.github.io/misa-framework/cpp-framework/standards/. The components of the *JSON* interface are briefly described here:

2.2.1. Application information

General information about a *MISA++* application can be either generated by the CLI or accessed from any *MISA++* processing output folder. It contains information like the application name, authors, version, description, license, organization, citation, and website. The standard also allows developers to easily cite dependency software packages.

2.2.2. Custom metadata

Metadata like quantification results that are attached to caches (see Section 2.1.2) are stored in a standardized location within the output directory. *MISA++* automatically documents the metadata in *JSON Schema* (http://json-schema.org/) format. Additionally, exported files contain information about the VFS location and data cache. This makes it possible to fully automatically extract quantification results from *MISA++* application output and link them to the data.

2.2.3. Parameters

Parameters are provided via a *JSON* file (see Section 2.1.3). Based on its name, type, and DAG node, each parameter is automatically assigned a unique location within the parameter file. It additionally contains information on how to link the VFS to locations on the hard disk (see Section 2.1.2). Users can either provide just the input and output directory, or have detailed control over each individual VFS entry.

Any *MISA++* application can automatically generate a human- and machine-readable documentation in *JSON Schema* format that outlines all possible properties of the parameter file. This includes name, description, data type, and default values of parameters defined in *C++* code. The documentation also contains a full description of the VFS and its caches, including information about its data

type, a description, and information about required metadata. The documentation therefore contains all necessary instructions on how to construct a valid parameter file, and information on which input data is required, and which output is generated.

3. Illustrative examples

In the following section, we briefly showcase applications of the *MISA++* framework. This includes integration of data types from third-party *C++* libraries, utilizing the *JSON* interface (see Section 2.2) for integration into third-party applications, and examples on optimizing algorithms via a re-implementation in *MISA++*.

3.1. OpenCV and OME TIFF integration

We developed a cache (see Section 2.1.2) that integrates the widely used image processing library *OpenCV* into *MISA++*. We also developed an extension that allows loading *OME TIFF* [11,12] image files into *OpenCV* data types.

To integrate *OpenCV* images, we developed a cache that automatically locates a compatible image file in its VFS directory. On accessing the image data, the file is de-serialized via OpenCV functions.

To integrate support for *OME TIFF* files, we use the *OME Files* library (https://www.openmicroscopy.org/ome-files/). Again, compatible files are detected if available and loaded via the *OME Files* library. Due to the structure of *OME TIFF* files, we implemented two cache types: (i) a cache that manages a single 2D plane, and (ii) a cache that represents the whole image file and provides access to each individual image plane cache. This allows to pass only a specific set of image planes to an algorithm. For example, a set of 2D planes can be distributed across multiple task nodes to parallelize processing (see Section 2.1.1).

3.2. ImageJ plugin

To showcase the capabilities of the *MISA++* interface, we developed a plugin for *ImageJ* that provides a graphical user interface for any *MISA++* application. It utilizes the human- and machine-readable documentation that can be generated by any *MISA++* application (see Section 2.2.3) to setup the user interface. The plugin depends on various libraries developed by third-party contributors (see Table S2).

The plugin maintains a customizable list of known *MISA++* applications and can automatically detect installed applications. By utilizing the standardized CLI, the plugin then queries the general application information, and the parameter schema. Based on this data, the plugin automatically derives information about input and output data, their VFS location, and which parameters are available — information necessary to execute an analysis.

On starting the *ImageJ* plugin, users are presented with a list of available applications. From within the application list, users have access to features that allow analyzing data with a single *MISA++* application, connect multiple applications into one pipeline, and analyze the output of an application. Step-by-step tutorials on how to use the plugin are available online at https://applied-systems-biology.github.io/misa-framework/imagej/step-by-step/. An in-depth feature showcase is given in the supplementary material.

3.3. Image analysis

To showcase the *MISA++* performance improvements, we developed three *MISA++* applications that combine common image analysis operations to process biological data as well as a single-operation benchmark. Our applications for biological image analysis are: (i) segmentation of glomeruli in 3D kidney data [13,14], (ii) segmentation of cells in phagocytosis assays [15], and (iii) application of deconvolution.

We compared our *MISA++* applications against equivalent *Python* and *Java* tools that use a DAG parallelization provided by *Snakemake* [16] and *Dexecutor* (https://dexecutor.github.io). Our *MISA++* implementations use *OpenCV* as image processing library, while the *Python* applications use *scikit-image* [17] and *NumPy* (https://numpy.org/). Our *Java* implementations are based on *ImgLib2* [18] and *ImageJ*. We enabled all available optimization functionalities, but also discuss the impact on disabling them in the supplementary material. All *C++* programs were compiled in *Release* configuration. All analyses were executed on a server with an Intel® Xeon® E7-8894 CPU, 1 TB of memory, and 30 threads.

To test if the different implementations produce the same results, we compared the distribution of characteristic readouts via a pairwise Bootstrap Kolmogorov–Smirnov test ($n_{boots} = 1000$). We consider two distributions to be significantly equal if their p-value $p \gg 0.05$.

Glomeruli analysis. Glomeruli are functional structures in kidneys. Their count and morphology are affected by various diseases and are therefore important factors to study. We analyzed LSFM images of whole murine kidneys (Fig. 3a) to quantify the glomeruli and extract information like their diameter and volume. The algorithm works as follows: After initial preprocessing, the kidney tissue is segmented. The resulting mask and preprocessed image are then used to segment glomeruli for each 2D image plane. Finally, the segmented 2D objects are reconstructed into 3D glomeruli, quantified and filtered (see supplementary material for details). The algorithm was initially developed in the *Python* programming language [13,14]. This implementation required approximately three days to analyze 23 murine kidneys with a total file size of 110 GB.

Our *MISA++* implementation supports *OME TIFF* files (see Section 3.1) that allow metadata like the voxel size to be stored within the image. We separated our implementation into two applications: (i) An application that segments kidney tissue, and (ii) an application that segments the glomeruli. The glomeruli segmentation tool re-uses the tissue segmentation code via its public API (see Fig. 3b). This design prevents code duplication, as tissue segmentation is a common part of any LSFM image analysis.

Input and output data are declared within the DAG root (see Section 2.1.2). The tissue segmentation tool expects an input *OME TIFF* file containing the input fluorescence image. It generates an *OME TIFF* image containing segmented tissue mask. We converted the *Python* algorithm into *C++* code and split into following tasks: (i) 2D tissue segmentation, (ii) tissue quantification, (iii) 2D glomeruli segmentation, (iv) 3D glomeruli reconstruction, (v) glomeruli quantification, and (vi) a task to filter false-positive glomeruli.

The DAG dependencies are set up so, that the tasks are executed in the listed order. Instead of running only one task at a time, the set of image planes is split across multiple instances of 2D tissue segmentation that run in parallel. The same applies to the 2D glomeruli segmentation.

The *MISA++* implementation was 95 times faster than the *Python* script and reduced the runtime from about 3 days to approximately 41 minutes (see Fig. 4a). See Listing S11 for the

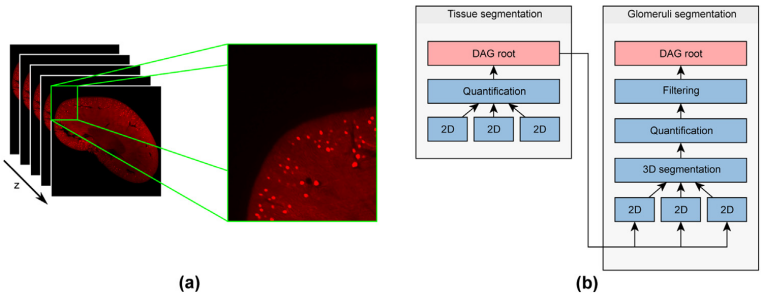


Fig. 3. The full kidney LSFM data set consists of hundreds of 2D image planes (a). Fluorescently labeled glomeruli are visible in the zoomed-in image. The image contrast was enhanced for visualization. (b) DAG structure of the *MISA++* glomeruli analysis tool. It consists of two applications, one responsible for segmenting the kindey tissue and another application solely responsible for segmenting the glomeruli. The tissue segmentation code is separated into its own *MISA++* application.

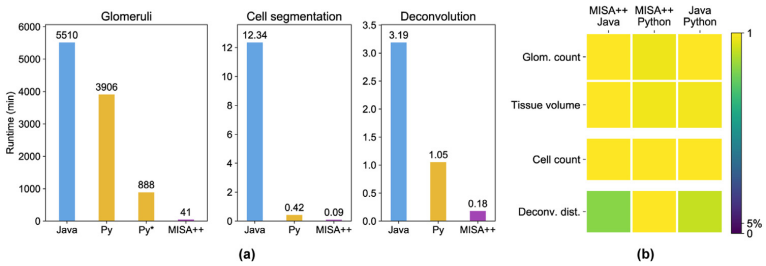


Fig. 4. Comparison between *MISA++*, *Python* (Py), and *Java* implementations. (a) The *MISA++* implementation is the fastest in all three test cases. *Snakemake* optimization of the original *Python*-based glomeruli segmentation (Py*) still did not achieve the speed of the *MISA++* implementation. (b) Pairwise testing for difference in distributions of the observed readouts lead to p-values well above the critical value of 5%.

parameter file. We updated the original *Python*-based implementation to use *Snakemake*. This reduced the runtime to about 14 h, which is still about 21 times slower than for *MISA++*.

Distributions of characteristic readouts were significantly equal between all implementations (see Fig. 4b and Table S3). We revealed differences in the glomeruli diameters of at most $0.42\ \mu m \pm 0.27\ \mu m$ (see Figs. S4 and S5), which is well below the image resolution with voxel size of $4.063\ \mu m \times 4.063\ \mu m \times 5\ \mu m$.

Cell segmentation and deconvolution. We describe the algorithms in more detail in the supplementary material. Our *MISA++* implementation was the fastest in both tasks. The cell count distribution as well as distributions in *Wasserstein distance* between standardized deconvolved and original image were significantly equal.

Single-operation benchmarks. *MISA++* was faster in five out of eight tested common image processing operations, which are explained in detail in the supplementary material, section 3.4 (see Fig. S8). Applying the Bootstrap Kolmogorov–Smirnov test, the Wasserstein distance between input and output images (see Table S4 and Fig. S9) are not significantly different, except in three operations where equivalent computations yield differences that can be explained by minor implementation details (see supplementary material).

4. Impact and conclusions

Our framework could be used to greatly improve the performance of existing methods. For example, the performance of the *ImageJ* implementation of our *Algorithm for Confrontation Assay Quantification* (ACAQ) to quantify host–pathogen interactions in images of endpoint experiments can be strongly increased [19]. Furthermore, the feature to integrate custom and third-party data types also allows applications beyond image segmentation, such as the implementation of our *Algorithm for Migration and Interaction Tracking* (AMIT) that monitors the dynamics of interacting cell systems by live cell imaging [20–22].

The *MISA++* framework combines the high performance of *C++* with easy-to-use applications by standardizing the development and interaction with *C++* programs. Developers can utilize our standardized components for parallelization, handling of third-party data types and quantification results, sample and algorithm parameters, creating a command line interface, and modularization into multiple applications and software libraries. Our standardized and powerful *JSON* interface allows any third-party program to easily interact with any *MISA++* application. We showcase our framework capabilities in our *OpenCV* and *OME TIFF* integration, our *ImageJ* plugin that automatically generates a graphical user interface for any *MISA++* application, and our high-performance re-implementation of the algorithm to segment glomeruli in LSFM data. By using our framework, the time to analyze a data set can be greatly reduced: Algorithms developed in user-friendly – but low-performance – software can be easily re-implemented as high-performance *MISA++* application. The advanced *JSON* interface then allows fully automatized interaction between those easy-to-use tools and the *C++* application for further data evaluation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Funding

This work was supported by the International Leibniz Research School for Microbial and Biomolecular Interactions Jena — ILRS Jena.

The authors would like to thank Sina Coldewey (Septomics Research Center, Jena University Hospital, Jena, Germany), and Matthias Gunzer (Institute for Experimental Immunology and Imaging, University Hospital Essen, Germany) for providing the LSFM images.

The authors would also like to thank Hanno Schoeler and Axel A. Brakhage (Leibniz Institute for Natural Product Research and Infection Biology — Hans Knöll Institute, Jena,Germany) for providing the images on phagocytosis assays.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.softx.2020.100405.

References

[1] Bradski G. The openCV library. Dr Dobb's J Softw Tools 2000.
[2] Yoo TS, Ackerman MJ, Lorensen WE, Schroeder W, Chalana V, Aylward S, et al. Engineering and algorithm design for an image processing API: a technical report on ITK-the insight toolkit. Stud Health Technol Inform 2002;586–92.
[3] Ragan-Kelley J, Barnes C, Adams A, Paris S, Durand F, Amarasinghe S. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In: ACM sigplan notices, vol. 48, no. 6. ACM; 2013, p. 519–30.
[4] Rueden CT, Schindelin J, Hiner MC, DeZonia BE, Walter AE, Arena ET, et al. ImageJ2: ImageJ for the next generation of scientific image data. BMC Bioinform 2017;18(1):529.
[5] De Chaumont F, Dallongeville S, Chenouard N, Hervé N, Pop S, Provoost T, et al. Icy: an open bioimage informatics platform for extended reproducible research. Nat Methods 2012;9(7):690.
[6] Berthold MR, Cebron N, Dill F, Gabriel TR, Kötter T, Meinl T, et al. KNIME: The konstanz information miner. In: Studies in classification, data analysis, and knowledge organization. Springer; 2007.
[7] Sommer C, Straehle C, Koethe U, Hamprecht FA. Ilastik: Interactive learning and segmentation toolkit. In: Biomedical imaging: from nano to macro, 2011 IEEE international symposium on. IEEE; 2011, p. 230–3.
[8] Carpenter AE, Jones TR, Lamprecht MR, Clarke C, Kang IH, Friman O, et al. CellProfiler: image analysis software for identifying and quantifying cell phenotypes. Genome Biol 2006;7(10):R100.
[9] Belevich I, Joensuu M, Kumar D, Vihinen H, Jokitalo E. Microscopy image browser: a platform for segmentation and analysis of multidimensional datasets. PLoS Biol 2016;14(1). e1002340.
[10] Haase R, Royer LA, Steinbach P, Schmidt D, Dibrov A, Schmidt U, et al. CLIJ: GPU-accelerated image processing for everyone. Nat Methods 2019;1–2.
[11] Goldberg IG, Allan C, Burel J-M, Creager D, Falconi A, Hochheiser H, et al. The Open Microscopy Environment (OME) Data Model and XML file: open tools for informatics and quantitative analysis in biological imaging. Genome Biol 2005;6(5):R47.
[12] Linkert M, Rueden CT, Allan C, Burel J-M, Moore W, Patterson A, et al. Metadata matters: access to image data in the real world. J Cell Biol 2010;189(5):777–82.
[13] Klingberg A, Hasenberg A, Ludwig-Portugall I, Medyukhina A, Männ L, Brenzel A, et al. Fully automated evaluation of total glomerular number and capillary tuft size in nephritic kidneys using lightsheet microscopy. J Am Soc Nephrol 2017;28(2):452–9.
[14] Dennhardt S, Pirschel W, Wissuwa B, Daniel C, Gunzer F, Lindig S, et al. Modeling hemolytic-uremic syndrome: in-depth characterization of distinct murine models reflecting different features of human disease. Front Immunol 2018;9:1459.
[15] Kraibooj K, Schoeler H, Svensson C-M, Brakhage AA, Figge MT. Automated quantification of the phagocytosis of Aspergillus fumigatus conidia by a novel image analysis algorithm. Front Microbiol 2015;6:549.
[16] Köster J, Rahmann S. Snakemake—a scalable bioinformatics workflow engine. Bioinformatics 2012;28(19):2520–2.
[17] van der Walt S, Schönberger JL, Nunez-Iglesias J, Boulogne F, Warner JD, Yager N, et al. the scikit-image contributors, scikit-image: image processing in Python. PeerJ 2014;2. e453. http://dx.doi.org/10.7717/peerj.453.
[18] Pietzsch T, Preibisch S, Tomančák P, Saalfeld S. ImgLib2—generic image processing in Java. Bioinformatics 2012;28(22):3009–11.
[19] Cseresnyes Z, Kraibooj K, Figge MT. Hessian-based quantitative image analysis of host-pathogen confrontation assays. Cytometry Part A 2018;93(3):346–56.
[20] Brandes S, Mokhtari Z, Essig F, Hünniger K, Kurzai O, Figge MT. Automated segmentation and tracking of non-rigid objects in time-lapse microscopy videos of polymorphonuclear neutrophils. Med Image Anal 2015;20(1):34–51.
[21] Brandes S, Dietrich S, Hünniger K, Kurzai O, Figge MT. Migration and interaction tracking for quantitative analysis of phagocyte–pathogen confrontation assays. Med Image Anal 2017;36:172–83.
[22] Al-Zaben N, Medyukhina A, Dietrich S, Marolda A, Hünniger K, Kurzai O, et al. Automated tracking of label-free cells with enhanced recognition of whole tracks. Sci Rep 2019;9(1):3317.

MISA++: a standardized interface for automated bioimage analysis - Supplementary text

Ruman Gerst[1,2], Anna Medyukhina[1] and Marc Thilo Figge[1,3,*]

[1]Applied Systems Biology, Leibniz Institute for Natural Product Research and Infection Biology – Hans-Knöll-Institute, Jena, Germany
[2]Faculty of Biological Sciences, Friedrich-Schiller-University Jena, Germany
[2]Institute of Microbiology, Faculty of Biological Sciences, Friedrich-Schiller-University Jena, Germany
[*]Corresponding author. Email address: thilo.figge@leibniz-hki.de

January 16, 2020

1 C++ framework

Library	Version	URL	Author/Citation
OpenMP	4.5	https://www.openmp.org/	[1]
Boost Libraries	1.67	https://www.boost.org/	Beman Dawes, David Abrahams, Rene Rivera; Boost Community
SQLite	3	https://www.sqlite.org/	SQLite Consortium
OME-Files	0.5.0	https://www.openmicroscopy.org/	[2, 3]
OpenCV	3.2.0	https://opencv.org/	[4]
JSON for Modern C++	3.5.0	https://github.com/nlohmann/json	Niels Lohmann
CMake	3.12	https://cmake.org/	Kitware Inc.

Table S1: C++ framework dependencies.

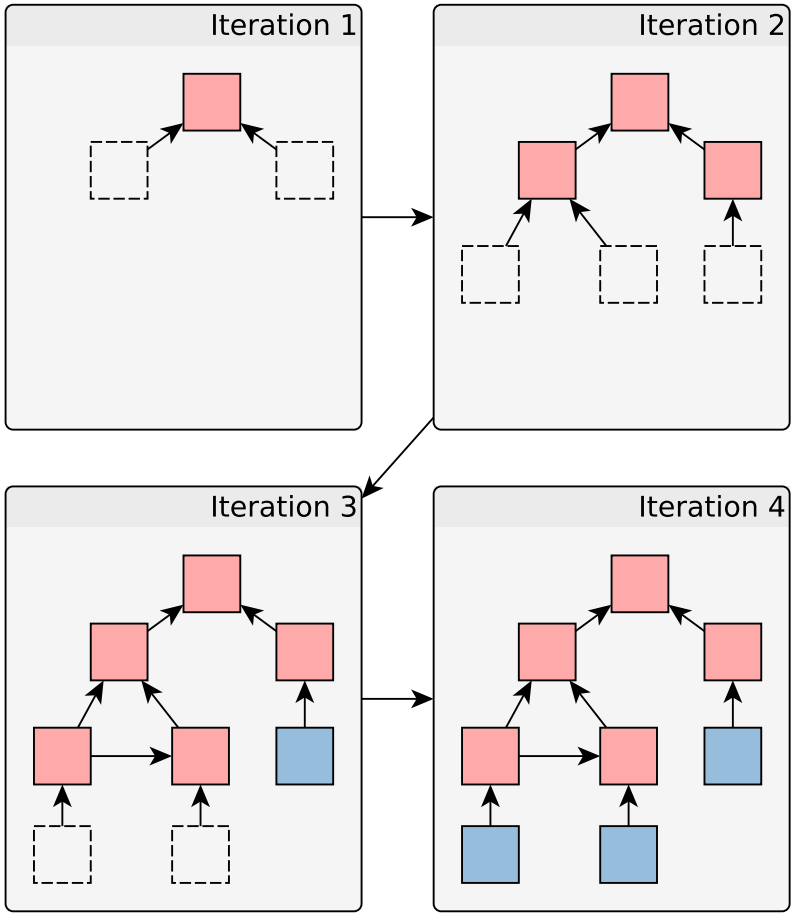


Fig. S1: *MISA++* organizes workloads in a DAG. It consists of two types of nodes: tasks (blue) and dispatchers (red). Dispatchers are responsible for creating the graph structure by iteratively instantiating direct neighbor nodes (white).

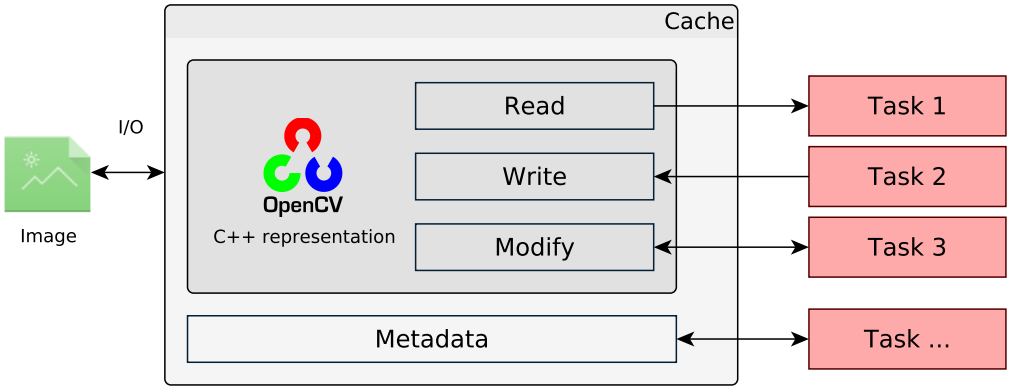


Fig. S2: Input- and output-data is organized in thread-safe and memory-efficient caches. Caches abstract data type specific input and output operations into reading, writing and modification. They additionally allow attachment of custom metadata.

1.1 Project development

Developing a project with *MISA++* involves creating a library that has a public *application programming interface* (API) that provides basic metadata, and a DAG root node. A separate project then wraps the DAG root node into an executable. This separation between executable and dynamically linked library makes it possible for other projects to re-use DAG.

A *MISA++* project consists of an project file for the *CMake* build system (see Listing S1), a package configuration file for *CMake* (see Listing S2), a global API function that returns metadata (see Listing S3), a C++ class that acts as data interface (see Listing S7), multiple node classes (see Listing S6), and a DAG root C++ class (see Listing S8). An example project structure can be found in Fig. S3.

Additional guides for project creation can be found online (`https://applied-systems-biology.github.io/misa-framework/cpp-framework/development/`).

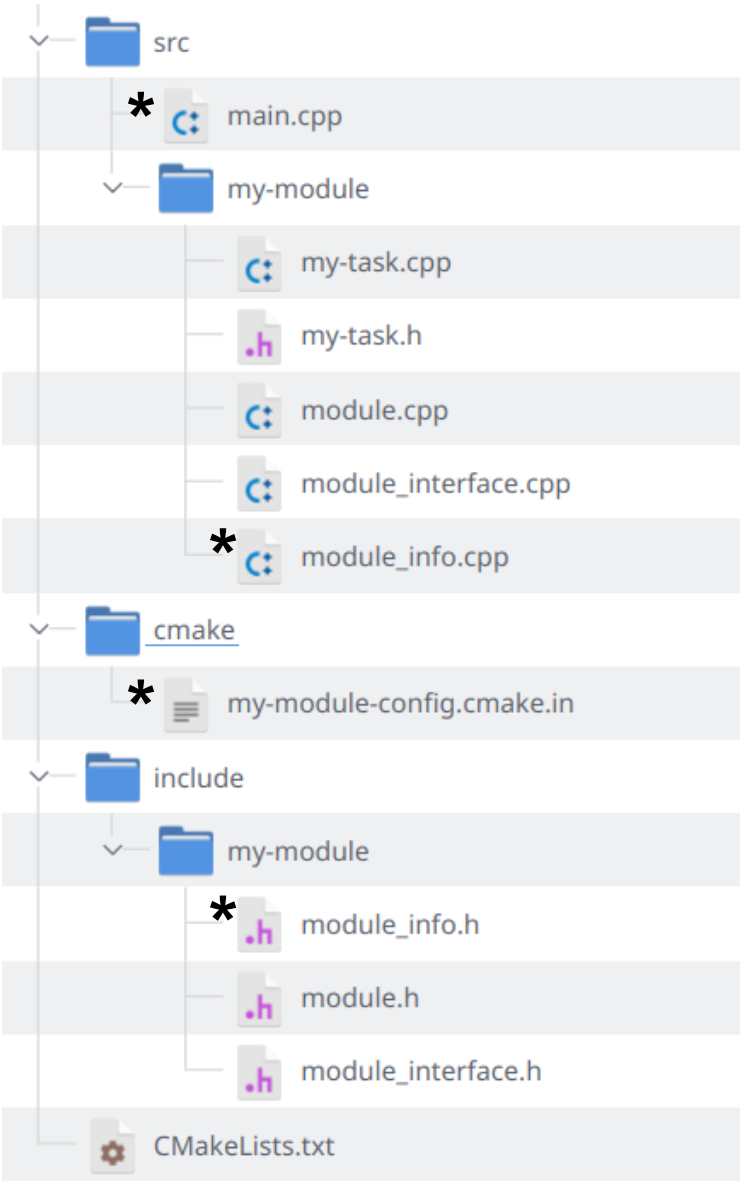


Fig. S3: Example project structure. We recommend a clear separation between API headers (folder `include`), and implementation code (folder `src`). The `CMakeLists.txt` project file contains instructions for the *CMake* build system. Our *MISA++ CMake* helper functions automatically generate files marked with an asterisk if they do not exist. The folder names (here `my-module`) correspond to the settings in the `CMakeLists.txt` file.

Project file The *CMake* configuration (see Listing S1) includes the code files into the build process, and sets up the installation operation that copies the generated library, executable, API headers, and additional metadata into the installation path. The *CMake* build system usually requires developers to manually set up the project configuration for API export. *MISA++* includes script helpers that are automatically available within the `CMakeList.txt` that includes the *MISA++* core library. Developers only need to provide some metadata and organize the project as shown in Fig. S3. The script then automatically introduces the necessary files and settings to allow proper installation and easy inclusion into other projects. This means that any additional files need to be initialized first, which is done by an *integrated development environment* (IDE), or manually by running following command in the project folder: `mkdir build; cd build; cmake ...`

```
cmake_minimum_required(VERSION 3.11) # Or higher if required
project(my-module VERSION 1.0.0 DESCRIPTION "")

find_package(misaxx-core REQUIRED)
find_package(misaxx-imaging REQUIRED)
# Add additional packages if necessary

add_library(my-module
include/my-module/module_interface.h
include/my-module/module.h
src/my-module/module_interface.cpp
src/my-module/module.cpp
src/my-module/module_info.cpp)

# Add additional link targets if necessary
target_link_libraries(my-module misaxx::misaxx-core misaxx::misaxx-imaging)

##
# MISA++ helper script (automatically included by Core Library)
##

set(MISAXX_LIBRARY my-module)
set(MISAXX_LIBRARY_NAMESPACE my_module::)
set(MISAXX_API_NAME my_module)
set(MISAXX_API_INCLUDE_PATH my-module)
set(MISAXX_API_NAMESPACE my_module)

# Automatically generate module_info() function if not existing
misaxx_with_default_module_info()

# Setup CMake for exporting the API (include files, etc.)
misaxx_with_default_api()

# Generate an executable out of the DAG root
# An appropriate main.cpp is added if not existing
misaxx_with_default_executable()
```

Listing S1: Example `CMakeLists.txt` project configuration file. It includes the source code into the build process and sets variables for the MISA++ CMake helper script.

Package configuration An example package configuration can be found in Listing S2. An automatically generated file is the *CMake* package configuration that contains information about dependency libraries for the compiled *MISA++* project. During the installation, it is copied to a directory that can be read by *CMake* to allow the library to be found with the easy to use `find_package()` function.

```
@PACKAGE_INIT@

# This should have the same find_package commands as
# the CMakeLists.txt file
find_package(misaxx-core REQUIRED)
find_package(misaxx-imaging REQUIRED)

# Automatically generated
if(NOT TARGET my-module)
```

```
include(${CMAKE_CURRENT_LIST_DIR}/my-module-targets.cmake)
endif()
```

Listing S2: Example *CMake* package configuration file. This file is required to allow other *CMake* projects to reuse the functions of the current project. It contains information about which dependency libraries *CMake* has to import.

Global metadata function Each *MISA++* should have a global function that returns information about the current software package (see Listing S3). To cite a third-party *MISA++* library, developers just have to add its metadata, which is accessible via the dependency API, to the current object. Non-*MISA++* software and papers can also be cited by manually creating a metadata object and setting its entries. An example for such advanced citing can be found in Listing S4.

```
// Function declaration in .h file
#include <misaxx/core/module_info.h>
#include <misaxx/imaging/module_info.h>

namespace my_module {
  misaxx::misa_module_info my_module::module_info();
}

// Function implementation - usually in a separate .cpp file
misaxx::misa_module_info my_module::module_info() {
  misaxx::misa_module_info info;
  info.set_id("my-module");
  info.set_version("1.0.0");
  info.set_name("my-module");
  info.set_description("");
  info.add_dependency(misaxx::module_info());
  info.add_dependency(misaxx::imaging::module_info());
  return info;
}
```

Listing S3: Example module information. The code defines a API function that returns metadata such as the name, version, or dependencies.

```
misaxx::misa_module_info misaxx::imaging::module_info() {
  misaxx::misa_module_info info;
  info.set_id("misaxx-imaging");
  info.set_version("1.0.1.0");
  info.set_name("MISA++ Imaging Support");
  info.set_description("Support for OpenCV");
  info.add_author("Ruman Gerst");
  info.set_license("BSD-2-Clause");
  info.set_organization("Leibniz Institute for Natural Product Research and Infection Biology - Hans Knoell Institute (HKI), Jena, Germany");
  info.set_url("https://asb-git.hki-jena.de/RGerst/misaxx-imaging/");

  // External dependency: OpenCV
  misaxx::misa_module_info opencv_info;
  opencv_info.set_id("opencv");
  opencv_info.set_name("OpenCV");
  opencv_info.set_url("https://opencv.org/");
  opencv_info.set_organization("OpenCV team");
  opencv_info.set_citation("Bradski, Gary, and Adrian Kaehler. \"OpenCV.\" Dr. Dobb's journal of software tools 3 (2000).");
  opencv_info.set_license("BSD-3-Clause");
  opencv_info.set_is_external(true);

  // External dependency: OpenCV
  misaxx::misa_module_info libtiff_info;
  libtiff_info.set_id("libtiff");
  libtiff_info.set_name("LibTiff");
  libtiff_info.set_url("http://www.libtiff.org/");
  libtiff_info.set_authors({"Sam Leffler", "Frank Warmerdam", "Andrey Kiselev", "Mike Welles", "Dwight Kelly"});
  libtiff_info.set_license("BSD");
  libtiff_info.set_is_external(true);
```

```
30
31    info.add_dependency(misaxx::module_info());
32    info.add_dependency(std::move(opencv_info));
33    info.add_dependency(std::move(libtiff_info));
34
35    return info;
36  }
```

Listing S4: Example module information with citations. The metadata objects allows developers to properly cite third-party software. The information can be later extracted via *MISA++* CLI commands. In this code example, we left out code already explained in Listing S3.

DAG nodes Each DAG node type corresponds to one C++ class. Task nodes (see Listing S6) have a method `work()` that contains the workload that should be executed by the node. From within a node, developers can access the data interface via the `get_module_as<T>()` function. Alternatively, as they are shared pointers, data caches can also be class attributes and assigned during node instancing.

Parameters are created by adding a class attribute of a *MISA++* parameter type. The object is then instantiated within a `create_parameters(builder)` function that connects the parameter attribute to a specific location within the current parameter file. This encapsulates raw access to *JSON* data to prevent errors, and allows the framework to extract the list of expected parameters via a simulated run.

Dispatcher nodes also have a `create_parameters(builder)` function, but specialized functions designed for DAG node creation instead of a `work()` method. As the DAG root is also a dispatcher node, see Listings S8 to S10 for examples how to assign workloads to dispatchers.

```
1  // Class declaration in .h file
2  #include <misaxx/core/misa_task.h>
3  #include <my-module/module_interface.h>
4  #include <opencv2/opencv.hpp>
5
6  namespace my_module {
7    struct my_task : public misaxx::misa_task {
8      using misaxx::misa_task::misa_task;
9      void work() override;
10   };
11 }
12
13 // Class definition in .cpp file
14 void my_module::my_task::work() {
15   std::shared_ptr<my_module::module_interface> module = get_module_as<my_module::
         module_interface>();
16
17   // RAII image data access
18   misaxx::readonly_access<cv::Mat> input_img_access = module->m_input_image.access_readonly();
19
20   // Threshold without unnecessary copy
21   cv::Mat thresholded { };
22   cv::threshold(input_img_access.get(), thresholded, 0, 255, cv::THRESH_OTSU);
23
24   // RAII output cache access
25   misaxx::write_access output_img_access = module->m_output_image.access_write();
26   output_img_access.set(std::move(thresholded));
27 }
```

Listing S5: Example task node. Data can be accessed through the module interface instance of the currently assigned sample. Data access is encapsulated through memory-efficient and thread-safe proxy objects. In this example, the input image of the current sample is thresholded with Otsu-thresholding. The resulting mask is written as output image.

```
1  // Class declaration in .h file
2  #include <misaxx/core/misa_task.h>
3  #include <my-module/module_interface.h>
4  #include <opencv2/opencv.hpp>
5
```

```
namespace my_module {
  struct my_task2 : public misaxx::misa_task {

    parameter<int> threshold;

    using misaxx::misa_task::misa_task;

    void work() override;
    void create_parameters(misaxx::misa_parameter_builder &parameters) override;
  };
}

// Class definition in .cpp file

void my_module::my_task2::create_parameters(misaxx::misa_parameter_builder &parameters) {
  // Register the parameter "threshold" with default value 50
  threshold = parameters.create_algorithm_parameter<int>("threshold", 50);
}

void my_module::my_task2::work() {
  std::shared_ptr<my_module::module_interface> module = get_module_as<my_module::
      module_interface>();

  // RAII image data access
  misaxx::readonly_access<cv::Mat> input_img_access = module->m_input_image.access_readonly();

  // Threshold without unnecessary copy
  // The parameter is queried using the threshold.query() function
  cv::Mat thresholded { };
  cv::threshold(input_img_access.get(), thresholded, threshold.query(), 255, cv::THRESH_BINARY
      );

  // RAII output cache access
  misaxx::write_access output_img_access = module->m_output_image.access_write();
  output_img_access.set(std::move(thresholded));
}
```

Listing S6: Example task node with parameters. Parameters are class attributes that are instantiated within the `create_parameters()` function. This allows *MISA++* to extract information about the parameters in a simulated run. The parameter value then can be queried via a `query()` method.

Data interface class The data interface class manages the data of one sample (see Listing S7). Its attributes are *MISA++* caches that either encapsulate input or output data of a specific type. The `setup()` function is used to link the caches to a subfolder in the sample input or output directory if no link has been already established by a potential parent data interface. Any cache can generate a self-description of its properties (for example file names or XML metadata) that can be passed from an input to an output cache.

```
// Class declaration in .h file
#include <misaxx/core/misa_module_interface.h>
#include <misaxx/imaging/accessors/misa_image_file.h>

namespace my_module {
struct module_interface : public misaxx::misa_module_interface {
  // Example: Input and output image
  misa_image_file m_input_image;
  misa_image_file m_output_image;

  void setup() override;
};
}

// Class definition in .cpp file
void my_module::module_interface::setup() {
  // The input image is loaded directly from the sample directory
  m_input_image.suggest_import_location(filesystem, "");

  // Create the output image within a "output" subfolder.
  // Copy metadata like filename from the input image
```

```
  m_output_image.suggest_export_location(filesystem, "output", m_input_image.describe());
}
```

Listing S7: Example module interface. The code defines the input and output caches and links them to locations within the *MISA++* filesystem. In this example, an input image and one output image are declared.

DAG root The DAG root, just like any dispatcher node, contains methods to instantiate a part of the DAG (see Listing S8). To allow the framework to automatically document the DAG without the need of data, the process is split into two parts: First, all possible tasks and sub-dispatchers are declared and assigned a name. Those "blueprints" are then instantiated in the second function. There can be as many instances of one blueprint as needed, allowing developers to easily model a parallel mapping (see Listing S9).

Any dispatcher (including the DAG root) have additional high-level capabilities of controlling the order of executed tasks (see Listing S10). They encapsulate two concepts: (i) A group of nodes, and (ii) a pipeline or chain of nodes. Developers can group together any set of nodes and model that another task is only executed after all grouped nodes have finished their workload. Chains allow easy creation of pipelines $(N_1, N_2, \ldots, N_{k-1}, N_k)$ where a node N_i is only executed after N_{i-1} has finished its workload.

```
// Class declaration in .h file
#include <misaxx/core/misa_module.h>
#include <my-module/module_interface.h>

namespace my_module {
  struct module : public misaxx::misa_module<module_interface> {
    using misaxx::misa_module<module_interface>::misa_module;

    void create_blueprints(blueprint_list &blueprints, parameter_list &parameters) override;
    void build(const blueprint_builder &builder) override;
  };
}

// Class definition in .cpp file
#include <my-module/my_task.h>

void my_module::module::create_blueprints(blueprint_list &blueprints, parameter_list &
    parameters) {
  // Register task "my-task" of type my_task
  blueprints.add(create_blueprint<my_task>("my-task"));
}

void my_module::module::build(const blueprint_builder &builder) {
  // Instantiate "my-task"
  my_task &task = builder.build<my_task>("my-task");
}
```

Listing S8: Example module DAG root. Tasks, dispatchers and sub-modules are first registered before their actual instantiation. In this example, a task `my_task` with the name `my-task` is declared without actually instancing it. Instantiation happens in the `build()` function where the task is instantiated based on its name. In more complex programs, multiple instances of the same task type can be created for parallelization.

```
// We assume that m_input_image contains a dynamic amount of slices
// We assume that m_output_segmented contains the same amount of slices as m_input_image

void my_module::module::create_blueprints(blueprint_list &blueprints, parameter_list &
    parameters) {
  // Register task "segment-2d-layer" of type segment_2d_layer
  blueprints.add(create_blueprint<segment_2d_layer>("segment-2d-layer"));
}

void my_module::module::build(const blueprint_builder &builder) {
  for(size_t i = 0; i < m_input_image.num_slices; ++i) {
    segment_2d_layer &task = builder.create<segment_2d_layer>("segment-2d-layer");
```

```
12      task.m_input = m_input_image.get_slice(i);
13      task.m_output = m_output_segmented.get_slice(i);
14    }
15
16    // The DAG contains now (m_input_image.num_slices) tasks of type segment_2d_layer
17    // Each task is responsible for segmenting one layer
18  }
```

Listing S9: Parallelization mapping example. Developers can instantiate any amount of nodes of the same type. This blueprint-like feature allows dynamic parallelization. In this example, we left out code that has already been explained in Listing S8.

```
1   // We assume that m_input_image contains a dynamic amount of slices
2   // We assume that m_output_preprocessed contains the same amount of slices as m_input_image
3   // We assume that m_output_segmented contains the same amount of slices as m_input_image
4   // We assume that preprocess_2d_layer has two cache attributes m_input_output
5   // We assume that segment_2d_layer has two cache attributes m_input and m_output
6
7   void my_module::module::create_blueprints(blueprint_list &blueprints, parameter_list &
        parameters) {
8     // Register task "segment-2d-layer" of type segment_2d_layer
9     blueprints.add(create_blueprint<segment_2d_layer>("segment-2d-layer"));
10
11    // Register task "preprocess-2d-layer" of type preprocess_2d_layer
12    blueprints.add(create_blueprint<preprocess_2d_layer>("preprocess-2d-layer"));
13
14    // Register task "quantify" of type quantify_task
15    blueprints.add(create_blueprint<quantify_task>("quantify"));
16  }
17
18  void my_module::module::build(const blueprint_builder &builder) {
19
20    // This group collects all segment_2d_layer tasks
21    group segmented_2d;
22
23    for(size_t i = 0; i < m_input_image.num_slices; ++i) {
24
25      preprocess_2d_layer &task_preprocess = builder.create<preprocess_2d_layer>("preprocess-2d-
            layer");
26      task_preprocess.m_input = m_input_image.get_slice(i);
27      task_preprocess.m_output = m_output_preprocessed.get_slice(i);
28
29      segment_2d_layer &task_segment = builder.create<segment_2d_layer>("segment-2d-layer");
30      task_segment.m_input = m_output_preprocessed.get_slice(i);
31      task_segment.m_output = m_output_segmented.get_slice(i);
32
33      // chain ensures that its entries are executed one after another
34      chain per_layer;
35      per_layer >> task_preprocess >> task_segment;
36
37      // add task_segment into the segmented_2d group
38      segmented_2d << task_segment;
39    }
40
41    // The >> operator ensures that the right operand is only executed when all tasks in the
          group are finished
42    quantify_task quantification = builder.build<quantify_task>("quantify");
43    segmented_2d >> quantification;
44  }
```

Listing S10: Dependency management example. Any dispatcher comes with powerful tools to further control DAG dependencies without the need of creating a separate dispatcher class. In this example, we left out code that has already been explained in Listing S8.

Library	Version	URL	Author/Citation
ImageJ, ImgLib2	2.0.0	http://imagej.net/	[5]
Gson	2.8.5	https://github.com/google/gson	Google LLC
Guava	26.0-jre	https://github.com/google/guava	Google LLC
JFreeChart	1.5.0	http://www.jfree.org/jfreechart/	Dave Gilbert
JFreeSVG	3.3	http://www.jfree.org/jfreesvg/	Dave Gilbert
flexmark-java	0.40.18	https://github.com/vsch/flexmark-java	Vladimir Schneider
Bioformats Plugin	6.0.0	http://www.openmicroscopy.org/bio-formats/	[2]
SQLite JDBC Driver	3.25.2	https://bitbucket.org/xerial/sqlite-jdbc	Taro L. Saito
Apache POI	4.0.1	https://poi.apache.org/	Apache Software Foundation
ICEpdf	6.2.2	http://www.icesoft.org/	Icesoft Technologies Inc.
Apache Commons Exec	1.4.0	https://commons.apache.org/proper/commons-exec/	Apache Software Foundation
Breeze Icons	-	https://github.com/KDE/breeze-icons	KDE e.V.
Font-Awesome-SVG-PNG	-	https://github.com/encharm/Font-Awesome-SVG-PNG	Damian Kaczmarek, Dominykas Blyžė

Table S2: ImageJ plugin dependencies.

2 ImageJ plugin

2.1 ImageJ plugin

To showcase the capabilities of the *MISA++* interface, we developed a plugin for *ImageJ* that provides a graphical user interface for any *MISA++* application. It utilizes the human- and machine-readable documentation that can be generated by any *MISA++* application to setup the user interface.

Step-by-step tutorials on how to use the plugin are available online in writing and video form at `https://applied-systems-biology.github.io/misa-framework/imagej/step-by-step/`.

Starting the *ImageJ* plugin opens a list of available *MISA++* applications along with a description. *MISA++* applications are either detected automatically or can be added by selecting the executable. From within the application list, users have access to features that allow analyzing data with a single *MISA++* application, connect multiple applications into one pipeline, and analyze the output of an application.

2.1.1 Data analysis

To analyze data, users must first setup the list of image samples. Samples are identified by their name and can be either imported from an existing folder that follows the VFS structure or provided via manual input. Images can be imported directly from *ImageJ* and are automatically converted into a format that is compatible with the cache. For non-image data types and custom caches, users can select the files from the file system. The *MISA++ ImageJ* plugin automatically creates the required input folder structure and warns users if data is missing. After setting up the data, users can change sample-, algorithm- and application-wide parameters via a graphical user interface. Documentation provided during the parameter declaration is accessible from within the interface. The settings are automatically validated by the plugin. Users can choose to either run the *MISA++* application on the current computer or export a data and parameter package for analysis on another computer or server. If the analysis is done on the current computer, the tool will offer to display and evaluate the results.

2.1.2 Pipeline creation

In addition to running a single *MISA++* application, our plugin also includes a tool to create pipelines. They are represented in a flowchart where processing steps are *MISA++* applications and arrows are dependencies. To create a pipeline, users can add any known *MISA++* application to the flowchart. Dependencies can be created by clicking a button that shows possible connections to the selected processing step. With an established dependency, output data can be passed from one step to the input of another step via the data import interface (see section S2.1.1). Another similarity to a single-application analysis is that pipelines can be either run directly from within *ImageJ* or exported for analysis on another computer.

2.1.3 Result analysis

We included a tool to simplify the further analysis of *MISA++* application results. It allows (i) importing result images back into *ImageJ*, (ii) browsing, summarizing, and plotting of quantification results, and (iii) analyzing the runtime. The graphical user interface displays all available output and input data. Compatible data can be imported back into *ImageJ*. The exact set of available actions depend on the cache data type and can be extended with custom plugins.

MISA++ applications store quantification results in a standardized format that can carry large amounts of additional metadata like units or localization information about a segmented

Algorithm	Tested value	MISA++ Java	MISA++ Python	Java Python
Glomeruli segmentation	Glomerular number	> 0.999	0.976	>0.999
Glomeruli segmentation	Tissue volume	> 0.999	0.978	0.983
Cell segmentation	Cell count	>0.999	>0.999	>0.999
Deconvolution	Distance	0.823	>0.999	0.912

Table S3: p-values calculated from pairwise comparing the characteristic algorithm readout distributions via Bootstrap Kolmogorov-Smirnov test ($n_{boots} = 1000$). All p-values are well above the critical value of 5%.

object. To allow efficient storage of the metadata, an external *MISA++* application transforms *JSON* into an SQLite (`https://www.sqlite.org/`) database. The user interface includes functions to easily query this database. Users can browse, filter, and export entries in table format. Tables can either be exported into commonly used spreadsheet formats, or further analyzed. We included features to extract of basic descriptive statistics, and plot the results directly from within *ImageJ*.

Any *MISA++* application is able to log its runtime with the option detailed per-task logs. Our *ImageJ* plugin comes with a tool to plot the runtime and extract descriptive statistics like the estimated multi-threaded speedup. Developers can use this information to spot performance bottlenecks and further increase the application performance.

3 Image analysis

3.1 Glomeruli analysis

```
{
  "filesystem": {
    "input-directory": "/data/input",
    "output-directory": "/data/output",
    "source": "directories"
  },
  "runtime": {
    "misaxx-ome": {
      "disable-write-buffer-to-ome-tiff": true
    },
    "num-threads": 30,
    "full-runtime-log": true
  },
  "samples": {
    "Kontrolle_Bonn 1a zoom063 z5 647_12-39-10": {},
    "Kontrolle_Bonn 1b zoom063 z5 647_13-22-21": {},
    "Kontrolle_Bonn 2a zoom063 z5 647_11-53-22": {},
    "Kontrolle_Bonn 2b_10-23-40": {},
    "Kontrolle_Ly6G M11a zoom08 z5 al647_14-26-05": {},
    "Kontrolle_Ly6G M11b zoom08 z5 al647_13-43-00": {},
    "Kontrolle_Ly6G M1a zoom08 z5 al647_12-38-06": {},
    "Kontrolle_Ly6G M1b zoom08 z5 al647_13-05-34": {},
    "Kontrolle_Ly6G M9a zoom08 z5 al647_11-36-12": {},
    "Kontrolle_Ly6G M9b zoom08 z5 al647_12-10-14": {},
    "d15 NTN 2a zoom08_15-33-00": {},
    "d15 NTN1a zoom08_15-19-34": {},
    "d15 NTN1b zoom08_13-43-09": {},
    "d15 NTN2b zoom08_14-23-01": {},
    "d15 NTN3a zoom08_16-04-45": {},
    "d15 NTN3b zoom08_16-29-01": {},
    "d7 NTN 1a zoom063 z5 647_09-40-46": {},
```

```
32      "d7 NTN 1b zoom063 z5 647_10-51-03": {},
33      "d7 NTN 2a zoom063 z5 488-647_15-46-09": {},
34      "d7 NTN 2b zoom063 z5_13-56-41": {},
35      "d7 NTN 3a zoom063 z5_17-01-52": {},
36      "d7 NTN 3b zoom063 z5 488- 647_12-11-22": {},
37      "d7 NTN2a neu_09-51-56": {}
38    }
39  }
```

Listing S11: Parameter file for *MISA++* glomeruli segmentation. The file sets up the VFS, number of threads, samples, and disables *OME TIFF* writing for consistency with the *Python* implementation.

Our glomeruli analysis algorithm [6, 7] applies the following steps: The initial preprocessing consists of a median filter that removes noise. Afterwards, the tissue is segmented via a percentile thresholding with subsequent morphological hole-closing operations. A filtering step is then applied to remove false-positive tissue regions via a mean intensity threshold. Glomeruli segmentation requires the minimum and maximum object radius and begins with segmenting the glomeruli in 2D. A morphological top-hat operation is applied to the preprocessed image to segment objects larger than the minimum radius. The glomeruli are then segmented via Otsu thresholding. Objects larger than the maximum size are removed via a morphological opening, while false-positive objects outside the tissue region are erased.

3.2 Cell segmentation

Segmentation and quantification of conidia – fungual spores – can be used to assess the behavior of immune cells [8]. One difficulty in segmenting conidia in 2D microscopy images is that cells might be merged during thresholding (see Fig. S6). A common method to separate merged cells consists of initial smoothing and Otsu thresholding. To separate merged cells, distance transformation is applied. The resulting image is used to find seed points via local maxima and as input for a watershed-algorithm.

The DAG is similar to the tissue segmentation where all 2D planes are segmented first and the resulting masks are accumulated afterwards.

Our *MISA++* implementation was over 137 times faster than the *Java* application and about 4.6 times faster than the *Python* implementation.

Our test for difference in distribution showed the results to be significantly equal in all tested cases (see Table S3).

3.3 Deconvolution

Deconvolution is a common image restoration technique (see Fig. S7). It is based on the idea that a microscopy image is convolved by a *point spread function* (PSF). If the PSF is known, an inverse filter operation in fourier space then can be used to restore the original image. In practice, naive inverse filtering yields usually bad results. There are different tools like *DeconvolutionLab2* [9] that implement alternative methods. For our example, we implemented the *Regularized inverse filtering* algorithm in *MISA++* and *Python*, and used *DeconvolutionLab2* for the *Java* implementation.

Given the fourier-transformation of the input image Y, the fourier-transformed deconvolved image X can be calculated via the *Regularized inverse filter* [9] method:

$$X = Y \cdot \frac{H}{H^2 + L \cdot \lambda \cdot L} \tag{1}$$

, where λ is the regularization factor, and L is the fourier-transformation of the 3×3 laplacian kernel. For our analyses, we set the regularization factor $\lambda = 0.001$.

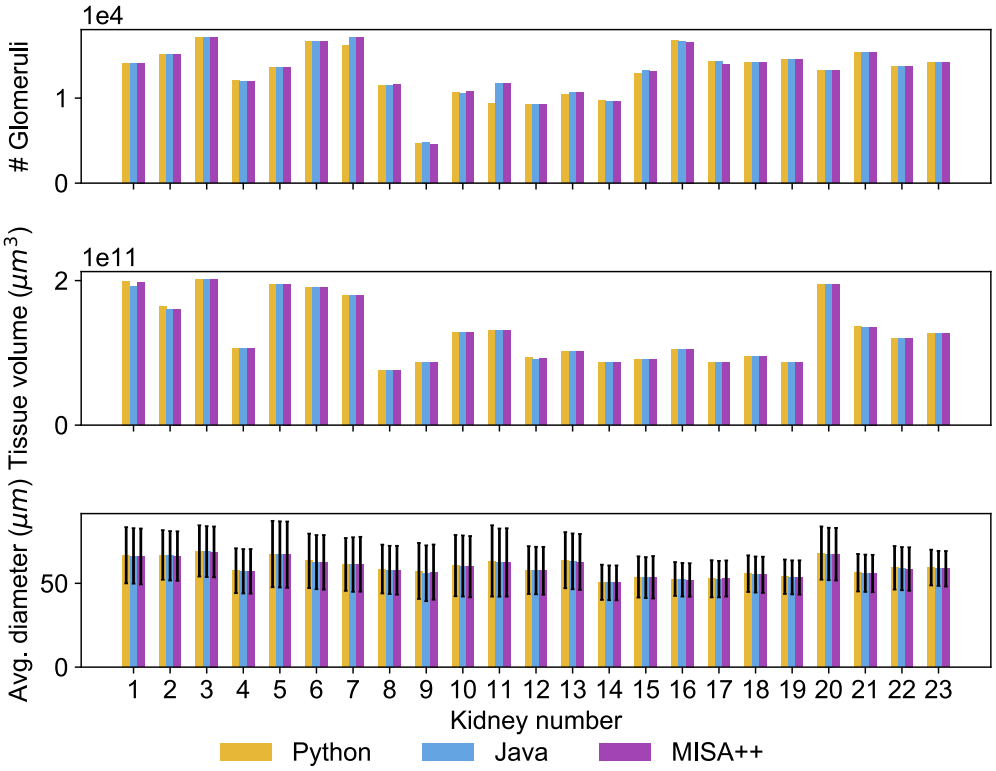


Fig. S4: Glomeruli segmentation result comparison. For the number of glomeruli, the maximum pairwise difference is 2344 (13.63%), and on average 168.58 ± 412.86 ($0.98\% \pm 2.4\%$). The tissue volume is at most $7.79 \cdot 10^9\,\mu m^3$ (3.85%) and on average $5.67 \cdot 10^8\,\mu m^3 \pm 1.32 \cdot 10^9\,\mu m^3$ ($0.28\% \pm 0.65\%$) different. The mean glomerulus diameter is on average $0.42\,\mu m \pm 0.27\,\mu m$ ($0.6\% \pm 0.39\%$) and at most $1.39\,\mu m$ (2.01%) different.

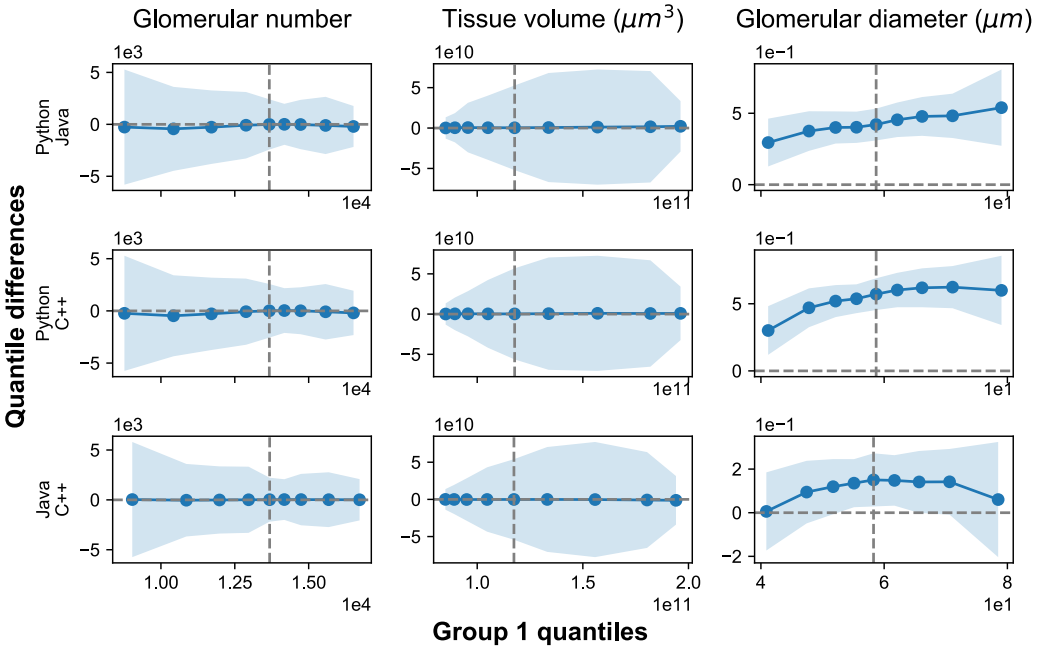


Fig. S5: Glomeruli segmentation result decile comparison. The zero-line is within the confidence intervals for glomerular number and tissue volume ($n_{boots} = 100$). The decile differences in diameters are outside the confidence intervals in all tested pairs, but well below image voxel size of $4.063\,\mu m \times 4.063\,\mu m \times 5\,\mu m$.

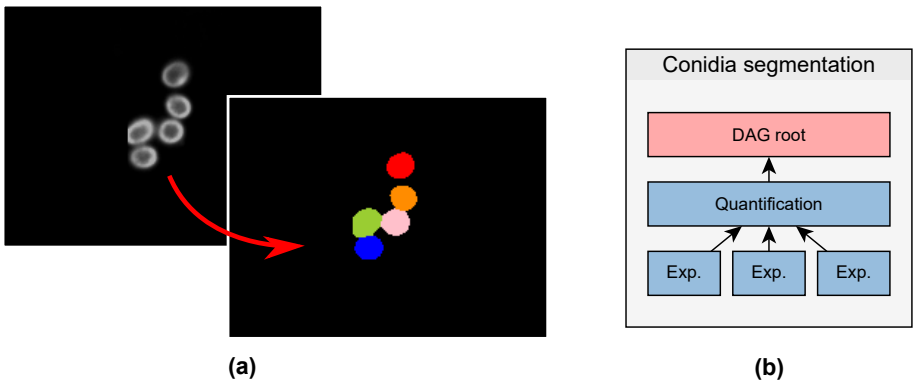


Fig. S6: Conidia (fungal spores) are fluorescently labeled (a). During thresholding, conidia might be merged into one object, so a watershed algorithm is used to separate them. The image contrast was enhanced for visualization. (b) DAG structure of the *MISA++* conidia segmentation tool. A sample consists of multiple 2D experiments that are segmented in parallel. Afterwards, the cells are quantified.

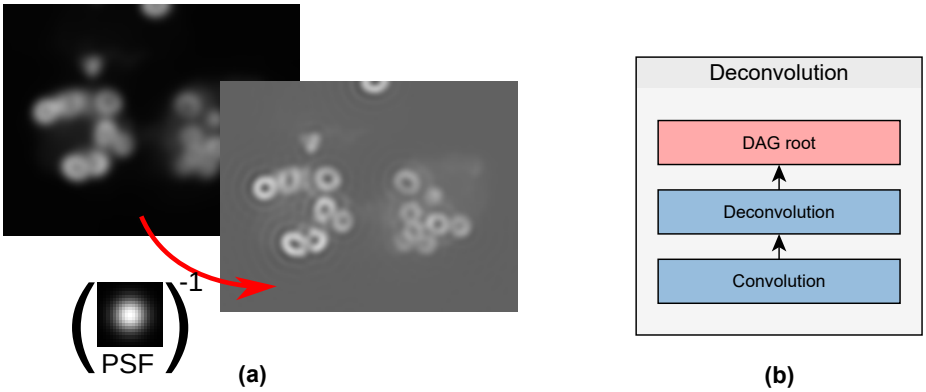


Fig. S7: (a) Convolution with a point spread function (PSF) can be theoretically undone with deconvolution. (b) DAG structure of the *MISA++* deconvolution test tool. An input image is first convolved given a known PSF and then deconvolved.

The DAG consists of a convolution task with a known PSF, followed by a simulated deconvolution with the same PSF.

Our *MISA++* implementation was about 17 times faster than the *Java* application and about 5.8 times faster than the *Python* implementation.

To allow comparison between different implementations, we calculated the *Wasserstein distance* between each deconvolved and its original image. Before calculating the distance, we first standardized the images to have a average value of zero, and a standard deviation of one. Each 2D image was converted to an 1D vector. Our test for difference in distance distribution showed the results to be significantly equal in all tested cases (see Table S3).

3.4 Single-operation benchmarks

The single-operation benchmark applications runs following tests:

canny	A canny edge detection algorithm $Canny(\sigma_{gaussian}, T_1, T_2)$ with gaussian preprocessing $\sigma_{gaussian} = 1$, and thresholds $T_1 = 0.1$ and $T_2 = 0.2$
fft-ifft	The input image is transformed into its fourier-space representation and transformed back into real space
io	Loading the input image and saving 7 output images as LZW-compressed 32 bit float TIFF files
median	Median-filtering with a 21×21 square mask
morphology	Dilation with a circle mask of radius 15
otsu	Otsu-thresholding
percentile	$65th$ percentile thresholding
wiener2	Applying a Wiener deconvolution according to the Matlab implementation (see `https://de.mathworks.com/help/images/ref/wiener2.html`)

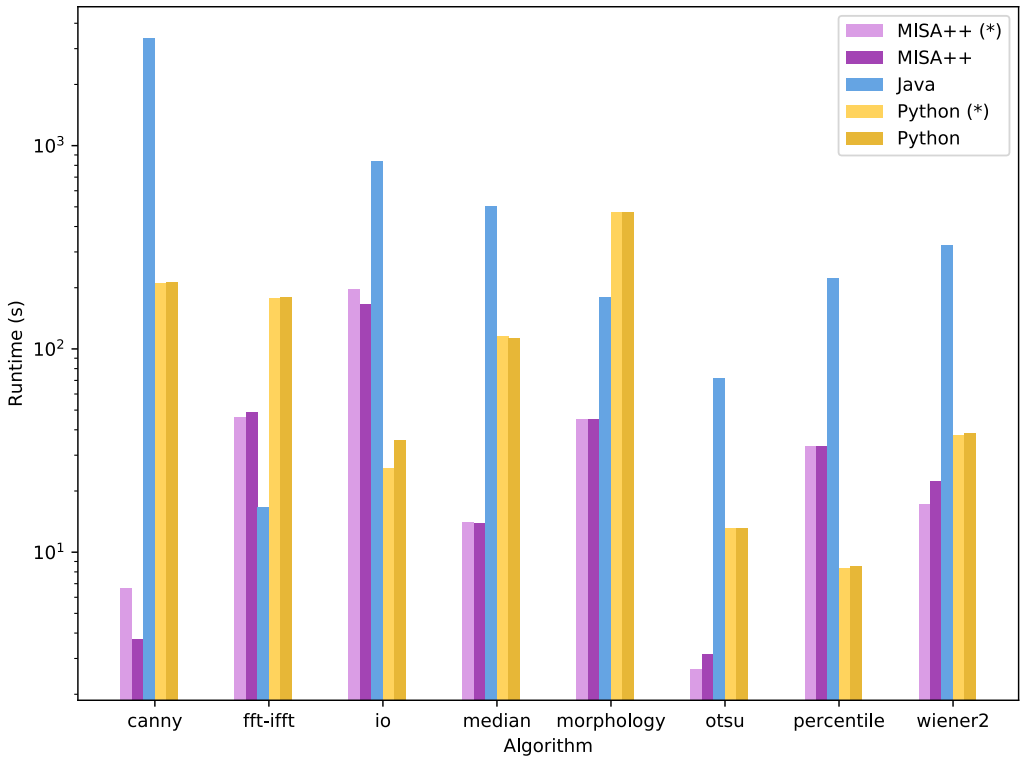


Fig. S8: Single-operation benchmark results. Results marked with a (*) were executed without additional parallelization offered by *OpenCV* and *Python* image libraries. *C++* was the fastest in 5 out of 8 cases.

Tested value	**MISA++ Java**	**MISA++ Python**	**Java Python**
Canny edge detector	0.807	> 0.999	0.957
Morphological dilation	> 0.999	> 0.999	> 0.999
FFT/iFFT	<2.22e-16	<2.22e-16	<2.22e-16
Median filter	<2.22e-16	> 0.999	<2.22e-16
Otsu threshold	> 0.999	> 0.999	> 0.999
Percentile threshold	> 0.999	> 0.999	> 0.999
Wiener deconvolution	<2.22e-16	<2.22e-16	> 0.999

Table S4: p-values calculated from pairwise comparing the distributions of the *Wasserstein distances* between original and convolved microbenchmark images. We uses a Bootstrap Kolmogorov-Smirnov test ($n_{boots} = 1000$).

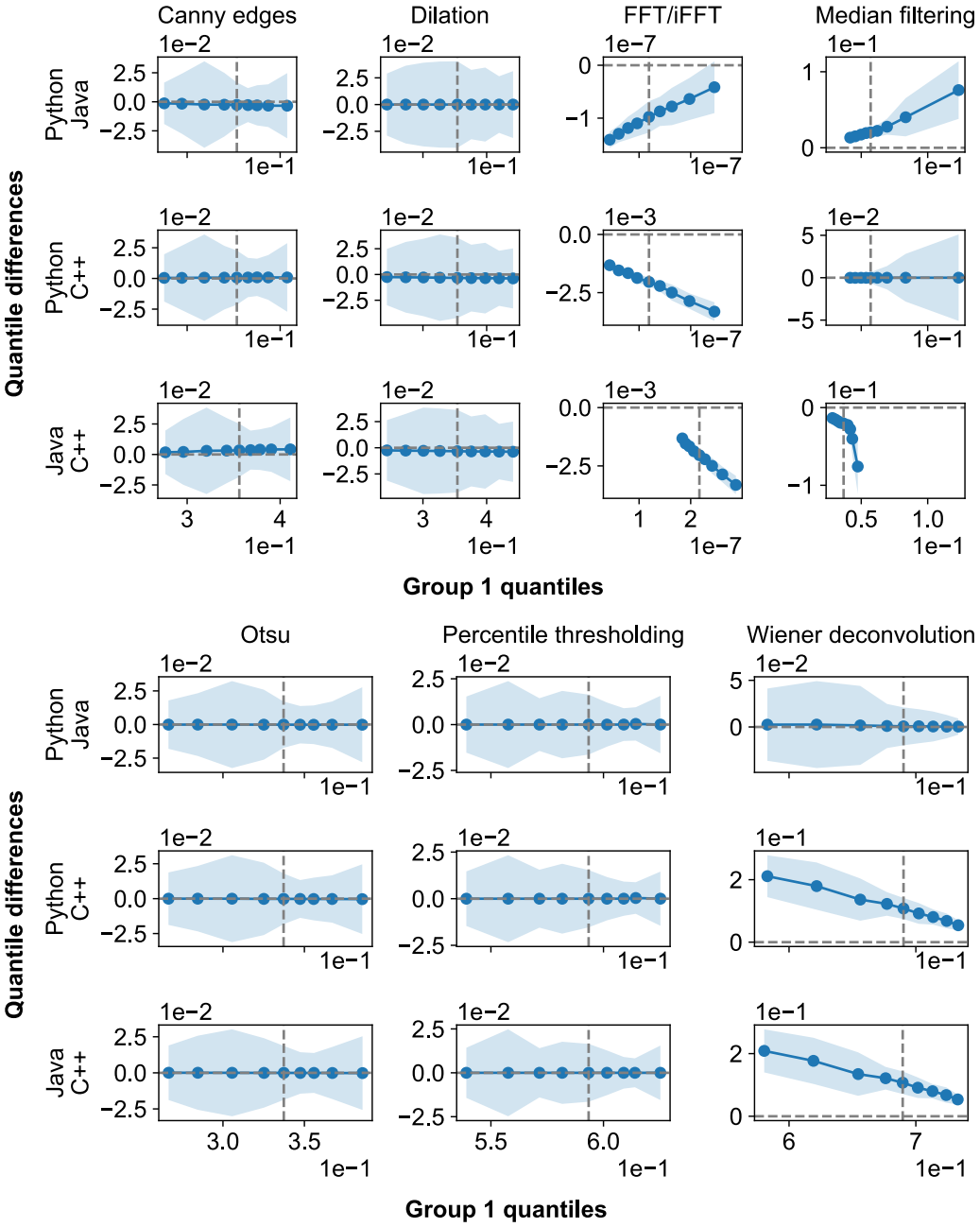


Fig. S9: Single-operation benchmark shift plots. All pairwise differences are within confidence intervals only in 5 out of 7 tested operations.

In our single-operation benchmark, *MISA++* was the fastest in 5 out of 8 cases. The cases where *C++* was not the fastest implementation were Fourier transformation, TIFF input and output, and percentile thresholding. We cannot explain the differences in TIFF IO performance, as both *MISA++* and *Python* use `libtiff` (`http://libtiff.org/`) for TIFF reading and writing with LZW compression enabled.

The Fourier transform performance can be explained by *Java* creating a differently sized frequency space representation: ImgLib2 offers *Fast Fourier Transform* (FFT) and *Inverse Fast Fourier Transform* (iFFT) algorithms, but does not allow developers to specify the size of the resulting frequency image. In our tests, ImgLib2 produced FFT outputs with smaller dimensions than the input image, while *OpenCV* and *NumPy* were instructed to generate outputs of the same size as the original image.

Lastly, the low *C++* in percentile thresholding performance can be explained by a naive custom implementation that first extracts all pixels and then applies a sorting algorithm. The performance can be optimized by applying partial sorting algorithms, as percentile extraction only requires a subset of conditions that are created by sorting.

As an addition to a versus-language comparison, we also tested the influence of additional optimization that is usually enabled by default for any image analysis library. Our results show differences between optimized and unoptimized workloads. The speed impact depends on the operation and the image library.

To allow testing if the results are equal across implementations, we calculated the *Wasserstein distance* between output and input image for each sample. We standardized the input image, so it has a mean value of zero and standard variance of one. We standardized the output images for non-thresholding algorithms (Dilation, FFT/iFFT, median filtering, Wiener deconvolution). For thresholding, we instead normalized the image by its maximum value. The reason behind this is that standardization transforms from binary value space to real value space, which distorts thresholded results.

We then compared the distributions via pairwise Bootstrap Kolgomorov-Smirnov test ($n_{boots} = 1000$). Our results show that not all distributions were significantly equal ($p \ggg 0.05$; see Table S4). Visualizations of the differences via decile shift plots are consistent with the test results (see Fig. S9). We address the observed differences below.

FFT/iFFT Evaluation of the result images revealed that the *C++* results were shifted, explaining the large difference between *C++* and other implementations. While the results are different, the program still applied a proper inverse FFT (iFFT) on previously generated frequency-space data. For real-world implementations (see Deconvolution example), the code is modified to extract the correct real image from periodic frequency space. We still see the performance comparison as valid, as equivalent operations were applied – just shifted by period.

While the difference between *Java* and *Python* results are significant ($p < 2.22\text{e-}16$), they decile differences are at most $2.7 \cdot 10^{-3}$. Considering implementation details like output sizing, and value scaling,

Median filter Our results indicate that the *Java* median filter implementation produces a different output. Further investigation revealed that *imglib2* converts the image to an 8-bit representation, while *C++* and *Python* implementations do not convert the floating point images. Although there are significant differences, the algorithms still have to apply similar operations for each neighborhood.

Wiener deconvolution The Wiener deconvolution results were significantly different between *C++* and the other implementations. We implemented this operation manually and re-confirmed that the operations are equivalent. To prevent zero-division errors, we added

a constant to to local variance component. This constant is very small (around 10^{-6}) and therefore can cause floating point errors depending on how the code is compiled into machine code.

3.5 Impact of additional optimizations

Python and *C++* image libraries allow developers to disable per-operation optimizations such as parallelization, or GPU computing. To test the influence of those optimizations, we repeated certain analyses without enabled optimizations. In case of *Java* image libraries, optimizations are usually enabled by default without an easy way for disabling them, or provided in code as explicit number of threads. Therefore, we did not compare the impact of optimization in *Java* applications.

Testing the impact of optimization on our three composite algorithms (see Fig. S10) shows that optimization can even have a negative impact if the workload is relatively small. For larger workloads, such as the 3D glomeruli segmentation, optimization can reduce the runtime by up to 38 minutes for the *Python* implementation. Optimization therefore is only reasonable if the processing steps are slow. Additional parallelization overhead, such as thread creation or uploading data to the graphics card, can negatively impact the performance for tasks that already run very fast.

Single-operation benchmarks (see Fig. S8) reveal that additional optimization impacts each operation in a different way.

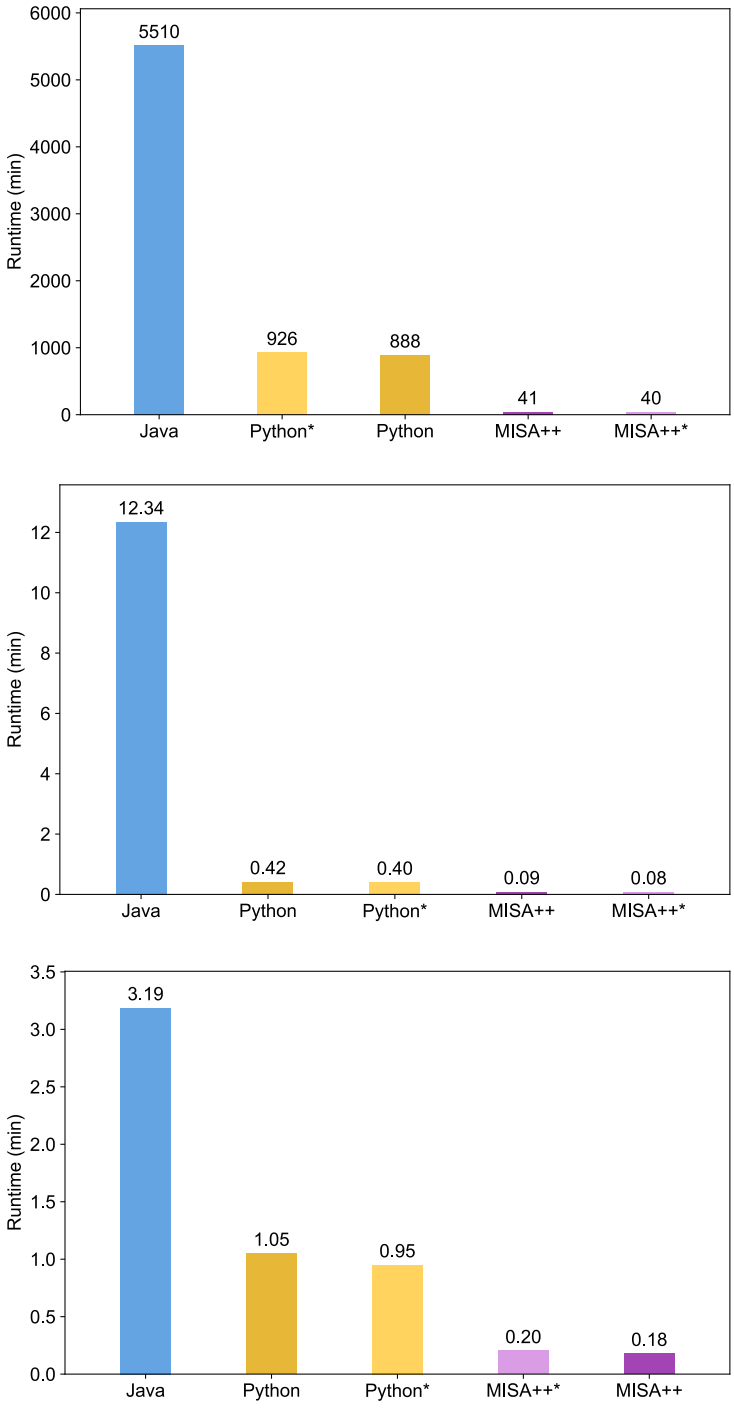


Fig. S10: Impact of additional optimization on composite algorithms. Results marked with an asterisk are run with disabled optimizations. The impact depends on the workload.

References

[1] L. Dagum, R. Menon, Openmp: an industry standard api for shared-memory programming, IEEE computational science and engineering 5 (1) (1998) 46–55.

[2] M. Linkert, C. T. Rueden, C. Allan, J.-M. Burel, W. Moore, A. Patterson, B. Loranger, J. Moore, C. Neves, D. MacDonald, et al., Metadata matters: access to image data in the real world, The Journal of cell biology 189 (5) (2010) 777–782.

[3] I. G. Goldberg, C. Allan, J.-M. Burel, D. Creager, A. Falconi, H. Hochheiser, J. Johnston, J. Mellen, P. K. Sorger, J. R. Swedlow, The open microscopy environment (ome) data model and xml file: open tools for informatics and quantitative analysis in biological imaging, Genome biology 6 (5) (2005) R47.

[4] G. Bradski, The opencv library, Dr. Dobb's Journal of Software Tools (2000).

[5] C. T. Rueden, J. Schindelin, M. C. Hiner, B. E. DeZonia, A. E. Walter, E. T. Arena, K. W. Eliceiri, Imagej2: Imagej for the next generation of scientific image data, BMC bioinformatics 18 (1) (2017) 529.

[6] A. Klingberg, A. Hasenberg, I. Ludwig-Portugall, A. Medyukhina, L. Männ, A. Brenzel, D. R. Engel, M. T. Figge, C. Kurts, M. Gunzer, Fully automated evaluation of total glomerular number and capillary tuft size in nephritic kidneys using lightsheet microscopy, Journal of the American Society of Nephrology 28 (2) (2017) 452–459.

[7] S. Dennhardt, W. Pirschel, B. Wissuwa, C. Daniel, F. Gunzer, S. Lindig, A. Medyukhina, M. Kiehntopf, W. W. Rudolph, P. F. Zipfel, et al., Modeling hemolytic-uremic syndrome: in-depth characterization of distinct murine models reflecting different features of human disease, Frontiers in immunology 9 (2018) 1459.

[8] K. Kraibooj, H. Schoeler, C.-M. Svensson, A. A. Brakhage, M. T. Figge, Automated quantification of the phagocytosis of aspergillus fumigatus conidia by a novel image analysis algorithm, Frontiers in microbiology 6 (2015) 549.

[9] D. Sage, L. Donati, F. Soulez, D. Fortun, G. Schmit, A. Seitz, R. Guiet, C. Vonesch, M. Unser, Deconvolutionlab2: An open-source software for deconvolution microscopy, Methods 115 (2017) 28–41.

Chapter 5

Discussion

Image-based systems biology (IbSB) enables the characterization of the interactions between host cells and pathogenic microorganisms, yielding knowledge that is useful for the treatment of life-threatening diseases, for example, sepsis [11], or hemolytic-uremic syndrome [11, 36]. This is achieved by a close collaboration between experimentalists and computer scientists: the characteristics of an experiment are captured into images that are quantified via advanced algorithms. The resulting measurements allow the estimation of invisible parameters and preliminary simulation of experiments *in silico*. Due to the complexity of interdisciplinary collaborations, it is essential to organize the storage of all generated data, including algorithms and their parameters, as well as the generated outputs. The basic principles behind digital research data management (RDM) are formalized in the concept of *Findability, Accessibility, Interoperability, and Reusability* (FAIR) [140]. Software can help to remove any existing barriers between experimentalists and computer scientists [90] and thus improve the adoption of FAIR, as well as new advanced technologies.

The goal of this thesis is to contribute towards the propagation of FAIR principles in the field of IbSB by the development of standardized, reproducible, high-performance, and accessible software for the quantification of interactions in biological systems.

The first publication of this thesis [59] (see section 4.1) introduced a reproducible and fully automated approach termed *MSOT cluster analysis toolkit* (*Mcat*) for the quantification of pharmacokinetics via multispectral optoacoustic tomography (MSOT). The algorithm was implemented as a plugin for *ImageJ* [115], thus making it accessible to non-programmers. In section 5.1 it will be discussed, how the software contributes towards the goal of this thesis, followed by perspectives on future developments. *ImageJ* is also subject of the second publication [47] (see section 4.2) that introduces a visual alternative to scripting termed *Java image processing pipeline* (*JIPipe*) into the popular platform [122]. The new software thus opens the development of advanced algorithms to non-programmers, facilitates the dialogue between computer scientists and experimental collaborators, as well as propagates the adoption of FAIR principles. A discussion is given in section 5.2 and will be about how the framework achieves these goals, as well as how it will be utilized and developed further. *JIPipe* was successfully ap-

plied in the characterization of the interactions between the soil-dwelling fungus *Mortierella verticillata*, its endosymbiotic bacteria, and the fungivorous nematode *Aphelenchus avenae* [24] (see section 4.3). Section 5.3 will discuss how automated image quantification pipelines contributed to the study, and how the approach can be enhanced for future publications. The final publication of this thesis [48] (see section 4.4) addresses the challenges of the increasing data volumes [111] by simplifying the development of high-performance and FAIR-compliant tools via a framework termed *modular image stack analysis for C++* (*MISA++*). The software package additionally introduces solutions for making these accessible to non-programmers. Section 5.4 will discuss the benefits of *MISA++*, how it attains easy accessibility and standardized data management, as well as how its feature set can be further enhanced.

5.1 Fully automated processing of MSOT data differentiates healthy from septic mice

Methods for assessing tissue functionality *in vivo* are of great interest in (pre-) clinical studies and ethical application of animal models. These allow to detect diseases that are correlated with a lower or higher function of cells in various organs. Examples are the reduction of kidney function in hemolytic-uremic syndrome [36], or the impact of sepsis to the liver tissue [141]. The current standard approach to obtain a quantification of organ function is by applying blood tests [145]. Yet, due to their indirect nature, it is not always possible to obtain enough information about the condition [23], leaving invasive biopsies as the only option. Another disadvantage is that no visualization of the whole organ is generated that would allow to isolate specific regions of an organ that show abnormal behavior. Ideally, the function of organ tissue is assessed non-invasively and in three spatial dimensions to enable the quantification on a per-region basis. MSOT is a non-invasive technique that allows to capture the functionality of tissue *in vivo* as four-dimensional (4D) data. Additionally, it supports multiple spectral channels that enable the tracking of multiple photoabsorbers at the same time. There are various clinically approved dyes available, including indocyanine green (ICG), Methylene blue, and Omocyanine [97]. The choice of the photoabsorber influences the type of tissue that is imaged. For example, the water-soluble molecule ICG is mainly eliminated inside the liver [136]. By utilizing MSOT to track the abundance of ICG in liver cells, the following pattern can be observed: at the point of injection, the signal increases due to the influx of the fluorophore from the blood stream. This is followed by a gradual decline in signal caused by degradation of ICG by liver cells. Consequently, this allows the comparison between healthy and damaged liver tissue via ICG kinetics.

In our publication [59] (see section 4.1), we compared the liver ICG uptake between healthy mice and animals that were injected with sepsis-inducing bacteria. Sepsis has a known influence on ICG processing of hepatocytes, thus making it an ideal candidate for evaluating our newly developed fully automated approach for quantifying MSOT data. This is compared to the performance of the current

standard approach that is based on the manual annotation of regions of interest (ROIs) by an expert. We could show that the commonly applied approach is highly sensitive to minor variances in the ROIs and does not yield the expected results, i.e. it could not differentiate between healthy and infected animals. The lack of an objective quantification method for MSOT data was the motivation behind our advanced computer algorithm termed *MSOT cluster analysis toolkit* (*Mcat*) that combines two machine-learning methods with the features of the *ImageJ* platform. Our approach comprises the suppression of movement artifacts, a fully automated segmentation of tissue regions via the deep neural network (DNN) *Cellpose* [126], and the unsupervised detection of regions within the animal tissue that show similar signal kinetics. These are quantified by the calculation of the per-cluster area under the curve. The combination of different machine learning operations makes use of their specific strengths. On the one hand, *Cellpose* can be easily adapted to various MSOT data sets. This is enabled via the training on the input of an expert, thus avoiding the introduction of abstract parameters that cannot be easily inferred from the MSOT setup. On the other hand, the second machine learning approach is entirely objective and considers only the pharmacokinetics as well as the number of expected groups, thus reducing the influence of expectancy bias.

To make our algorithm accessible to non-programmers, *Mcat* is implemented as plugin for *ImageJ*. We decided to base our implementation on the *Java* programming language instead of utilizing the existing scripting capabilities of *ImageJ*. While the benefit of these integrated programming languages include the simplified development and maintenance, scripts only have limited capabilities to create graphical user interfaces (GUIs). This makes it difficult to develop a user-friendly interface around the various parameters of our *Mcat* algorithm. Additionally, there are various image analysis operations that cannot be easily implemented inside a script, including K-means clustering, integration of *Cellpose*, custom statistics, and the z-transform algorithm. Thus, we decided to develop a *Java* plugin that comprises more code compared to a macro, but allowed us to develop a purpose-built GUI with integrated project, parameter, and data management. As *ImageJ* functions are available from within *Java*, we could combine these with our custom implementations.

To summarize, our publication makes the quantitative analysis of MSOT data more reproducible and accessible, thus facilitating the adoption of this technique for future studies and application in clinical environments.

Perspectives

Extending the spatial analysis to all three dimensions. Although MSOT setups are capable of generating images with three spatial dimensions, the current implementation of our software can only analyze two-dimensional (2D) time series. As researchers are required to select a depth, our software thus introduces an additional manual review step that is subject to user bias and has the potential of significantly impacting the quantification results. For example, it is possible that the user selects a slice with relatively low or high activity. Due to the loss of information, it is not possible to capture the pharmacokinetics of a whole organ.

To enable the analysis of three-dimensional (3D) data, the algorithms utilized by our software need to be adapted to the third spatial dimension. The first challenge is the modification of the image registration and smoothing methods that internally combine the frames of the 2D time series into a 3D image as expected by the registration and smoothing algorithms. On analyzing a 3D time series, the result is consequently a 4D image that cannot be processed by the current choice of registration operators. A solution is to apply the preprocessing steps on a per-slice basis, i.e. each depth of the 3D time series is processed individually. Another option is to replace the preprocessing algorithms by alternative implementations that are capable of handling three spatial dimensions, e.g., algorithms provided by the *3D ImageJ Suite* [102].

To adapt the segmentation of tissue to 3D images, the existing implementation can be executed on a per-slice basis. The disadvantage of this strategy is that image operations are not aware of the 3D-neighborhoods, which can result in artifacts at the segmentation borders. These then would need to be corrected with additional 3D-aware operations, for example, a 3D morphological operation. Alternatively, all operations contained in the pipeline can be adapted to three dimensions. *Cellpose*, which provides the initial probability map already supports 3D segmentation. Thresholding, morphological operations, and additional image processing steps can be replaced by their 3D counterparts and are either already available via native *ImageJ* functions, the *MorphoLibJ* [78] library, or the *3D ImageJ Suite*.

The clustering of 3D pixels can be trivially implemented by adapting the existing feature vector generation method to add all voxels within a 3D image. Thus, the clustering operations and all related statistical methods can be applied as-is. Finally, *Mcat* generates a visualization of the cluster assignments that can be easily adapted from 2D to 3D.

Open source web-based analysis tool for clinical use. To make *Mcat* suitable for a clinical environment, there are still two major points where improvements must be made: first, our tool should allow manual intervention via semi-interactive analysis; second, the access to the program must be simplified and adapted to as many devices as possible, including low-end computers, phones, and tablets. Currently, *Mcat* executes all its steps automatically, which is sufficient for the analysis of animal models. For clinical applications, medical doctors must be provided with means to review the intermediate results and apply corrections if needed. There is currently no option to generate one or multiple preliminary tissue ROIs that can be revised in an interactive interface, and no mode to test and modify the behavior of the clustering algorithm.

Due to the modular design of *Mcat*, interactive settings can be easily implemented as optional GUI that is opened during the analysis. A disadvantage of this approach is the missing separation between computing and GUI components, which makes it difficult to run our software on dedicated compute servers that do not provide a graphical interface. The only option left is to upgrade office computers to a level that is capable of running *Mcat*, which is expensive and limits the ease of access. A solution to this accessibility issue is to separate the computational and

GUI components of *Mcat* into dedicated software packages with different roles: one program acts as server that runs workloads on high-performance hardware, while a dedicated web application acts as GUI that runs on a phone, tablet, or computer. The communication between the server and client is established via standardized application programming interfaces (APIs) based on *Representational State Transfer* (REST) and *Javascript* object notation (JSON). The server component can be implemented in *Java*, thus allowing the re-use of existing *Mcat* functions. For the development of the web application, existing frameworks can be utilized, for example, *React* (Meta Platforms, Inc.), or *Angular* (Google LLC). The benefit of web applications is that they can be utilized both in a remote and local environment in form of websites that can be packaged into applications via the *Electron* framework (GitHub Inc).

A challenge with our web-based software is that five-dimensional images generated by MSOT are large, thus making it difficult to visualize these on low-end devices due to limited memory. A solution involves the implementation of an API that enables the request of individual slices and low-resolution images. For example, unless the user requests a high-resolution image, only a small preview is loaded from the server. An alternative technique is the integration of the image database *OMERO* [5], which already provides ready-to-use solutions for the transfer and visualization of large data sets via web interfaces. An additional benefit of *OMERO* is that the MSOT setup can upload data directly to the image database. *Mcat* can be also integrated directly into the *OMERO* platform, due to its support for plugins, thus removing the need for a custom web interface implementation and providing users with a uniform interface for processing and integrating various types of data.

A standardized web-based platform for data management and processing also opens the possibility to an extension outside the field of MSOT via the integration of additional patient-related data, for example, endoscopy or blood test results. A centralized storage allows the collection of long-term comparisons and statistics that enable the generation of predictions via computer models with the purpose of assisting medical doctors in selecting an appropriate treatment. As the software is open source, servers can be set up within local infrastructure, ensuring secure data storage according to data protection laws.

Implementation of alternative image processing operators for future projects. The current version of our *Mcat* implementation makes specific choices on the algorithms that are responsible for the preprocessing, tissue analysis and clustering steps. The consequence is that the tool cannot be easily adapted to scenarios where the current selection of operators yields unsatisfactory results. Thus, alternative algorithms will be briefly discussed in the following paragraphs.

Images are first processed by the registration method *MultiStackReg* [129] to correct for the movement of the animal. While this method was successful for our data set, other experiments might require the usage of alternative methods, for example, *bUnwarpJ* [6], *Fijiyama* [44], or implementations of scale-invariant feature transform [85]. Registration methods cannot fully correct for breathing artifacts.

Mcat thus applies an arithmetic averaging over the time axis. The currently available parameter set only allows the selection of the filter size. To make our approach adaptable to future experiments, it can be extended by a choice of various smoothing functions, including median and Gaussian blurring, and the option to insert custom mathematical expressions. Implementations of alternative registration and smoothing methods are available for *ImageJ* or can be included by utilizing existing *Java* libraries.

Another limitation with the current *Mcat* software is that the fully automated tissue segmentation relies exclusively on the *Cellpose* [126] DNN architecture to obtain the tissue probabilities. A challenge of MSOT data analysis is that the parameters of the tomography device greatly influence the image generation. Thus, it is expected that our pre-trained model performs worse on data sets from other experiments. To resolve this issue, researchers could be encouraged to share generated models in an online repository. This can be achieved by collaboration with the *BioImage Model Zoo* (`https://bioimage.io/`) repository that is already successfully utilized in the *DeepImageJ* plugin [89]. *Mcat* can be extended to make use of the *BioImage Model Zoo* by generalizing the segmentation functions into a standardized library with the option of multiple backends. This abstraction of implementation details is required, as the *Pytorch* [106] models that are generated by *Cellpose* are incompatible with *Tensorflow* [1] models utilized by *DeepImageJ*. The general applicability opens the possibility to create new backends via the *ImageJ* plugin interface, including algorithms based on classical image analysis and alternative machine learning techniques.

Currently, our software exclusively utilizes a K-means clustering method, which is known to not always yield optimal results depending on the properties of the data [87, section 20.1, page 288]. Users should be provided with an extensible set of alternative clustering methods, for example, clustering via a Gaussian mixture model [87, section 22.2], a clustering based on DNNs, or an algorithm that makes use of a support vector machine [99]. The disadvantage of K-means and the other mentioned methods is that they find a predefined number of partitions, meaning that assumptions are made on the number of clusters. An alternative is to implement a hierarchical clustering approach that does not create one partitioning, but instead a tree of nested clusters [87, section 20, page 284]. This representation can be used to generate outputs that visualize the detected regions interactively via a generated website. This can aid medical doctors in clinical applications where experts need to be provided with alternative solutions. As with the generation of flat clusters, there are various algorithms available that generate a hierarchy of clusters. For example, there are approaches that are based on the iterative merging of clusters until a single tree is formed [96], including nearest-neighbor clustering, complete linkage, and methods based on the calculation of weighted and unweighted group averages.

Generalization into a software framework. Currently, *Mcat* provides a custom GUI, a command line interface (CLI) for execution in server environments, as well as an API for executing *Mcat* algorithm and managing data and parameters. The single purpose of our software to analyze MSOT data thus reveals a bottleneck

with the development of related applications: a software that utilizes an entirely different set of image processing operations will still require the implementation of functions to manage data and parameters, as well as interfaces for users and developers. To reduce the development time, a new analysis tool can be based on the code of an existing application, which has the drawback of complicating its maintenance: improvements must be communicated between multiple source code repositories, which is difficult with the increasing number of projects and changes that accumulate over time. Thus, it is reasonable to outsource shared functionality into a dedicated framework library that provides a standardized API for other programs. This simplifies the creation of software tools, reduces their code size, and centralizes the maintenance of common features. A central component of the framework can be a plugin system, as implemented in *ImageJ*, thus enabling the dynamic addition of features by third-party software libraries.

Mcat already contains components that can be adapted to a framework, including the data management and parameter system, the project format, and the interfacing between processing steps that allows alternative algorithms. One major drawback of the developer-focused design is that only programmers are able to fully customize the behavior of a tool. While there are already options to modify the behavior of specific algorithms via parameters, it is impossible for non-programmers to add custom steps into the workflow or re-arrange the order of steps within *Java* code. A more accessible alternative is to redesign *Mcat* as set of modules with standardized inputs and outputs that can be freely connected into a pipeline via a GUI. These modules would be provided by our software, but also automatically include the set of functions provided by *ImageJ*. The implementation of such an accessible framework is the subject of our second publication [47].

5.2 Visual programming opens the development of fully automated pipelines to non-programmers

In our last manuscript [59] the implementation of our algorithm was based on the widely popular [122] *ImageJ* platform. The hallmark of the latter is its user-friendly GUI for the interactive execution of image analysis operations. Yet, advanced features that are required to design FAIR-compliant [140] software are only accessible via a macro scripting language. This includes the design of advanced and reproducible algorithms that are formed by the application of multiple image processing operations, the implementation of scalable analysis pipelines, and the adoption of standardized data storage modes. The required knowledge to understand and develop macros is not widespread in the community of researchers with a background in life sciences, thus contributing towards a gap between computer scientists and experimentalists [90]. This circumstance is in particular detrimental to the implementation of IbSB that relies on interdisciplinary collaborations and mutual understanding of all involved computational and experimental methods. Thus, it is paramount to present an easily accessible alternative to scripting that provides access to the development of advanced, reproducible, and FAIR-compliant image analysis workflows to any researcher.

An alternative to macro programming that facilitates the communication of computer algorithms can be found in a visual programming language (VPL) [75]. This concept is based on a visual representation of an algorithmic pipeline as user-modifiable flow chart, while technical details are automatically maintained by the language itself, e.g., the correct order of operations and data management. VPLs thus circumvent the necessity of mastering a multitude of abstract programming concepts, including programming structures, pointers and references, function parameters, complex data types, error handling, and software libraries [107]. The simplicity and general applicability of VPLs is highlighted in variety of open source and commercial implementations that include *Scratch* [88], *KNIME* [14], *Icy* [35], *CLIJ* [55], *Galaxy* [66], and *Apeer* (Carl Zeiss Microscopy GmbH). Specifically *Icy* and *KNIME* are of interest for the design of image analysis pipelines, as these open source tools allow the visual combination of image processing operations. Yet, an equivalent functionality is absent in the widely popular [122] *ImageJ* platform.

In our publication [47], we introduced the *Java image processing pipeline* (*JIPipe*), which is a plugin for *ImageJ* that encapsulates its existing functionality into a VPL with the goal of providing an alternative to macro programming for beginners and advanced users. Due to the visual nature of *JIPipe* workflows, it is straightforward to communicate the individual steps involved in an image analysis pipeline to an experimentalist, thus promoting an understanding of the capabilities and limitations of image processing approaches. A VPL also simplifies the adaption of existing image analysis pipelines by a life scientist for the efficient applications of preliminary analyses, thus closing the gap between computer scientists and experimentalists.

JIPipe aims to provide a viable alternative to the writing of macro code. This goal can only be achieved, if our tool fulfills four conditions: first, our software must encompass the features of *ImageJ*; second, *JIPipe* must be more accessible compared to macro programming; third, our software must be easy to learn for users unfamiliar with macro programming; and fourth, our language must be capable of applying batch processing. These points will be briefly discussed in the following paragraphs.

The basis of achieving a feature parity between *ImageJ* and *JIPipe* is the adaption of the GUI-focused *ImageJ* data handling concept into data objects that can be handled by a VPL. These objects are built around the existing *Java* APIs provided by *ImageJ* and thus allow the easy interfacing with existing *ImageJ* operations. A challenge with *ImageJ* is that it comes with two APIs that manage image processing commands: the older *ImageJ1* API, and a modern and more standardized implementation based on *ImageJ2*. For our publication, we decided to implement algorithms of both APIs, although via of two different strategies. Due to the lack of standards regarding parameters and data management, *ImageJ1* commands were encapsulated on an individual basis, meaning that each function corresponds to one or multiple program code files in the *JIPipe* source code. This allowed us to utilize our advanced parameter and documentation system to generate user-friendly variants of the existing algorithms. For example, we integrated documentation that is otherwise only available within the original program code and thus inaccessible to non-programmers. The integration of *ImageJ2* was simplified by its high stan-

dardization that allowed the development of a fully automated encapsulation of most *ImageJ2* functionality. A disadvantage of this automated approach is that its success relies on the layout of the *ImageJ2* algorithm. For example, various commands cannot be adapted due to incomplete or conflicting parameter definitions. Additionally, we developed *JIPipe*-exclusive functions that cover automated file system operations, image reading, metadata generation and management, visualization, and plotting. As *ImageJ* is a well-established software with hundreds of plugins, we were not able to encapsulate all their functionality into dedicated nodes. To support these missing commands, we designed nodes for executing *ImageJ* macros and *Jython* [67] scripts. As these languages have access to all features of *ImageJ* and its plugins, *JIPipe* thus achieves full feature parity with *ImageJ*. While programmable nodes rely on script languages and thus are less accessible, the necessary code can be generated via the "macro recorder" feature of *ImageJ* that converts GUI operations into an equivalent code.

As *JIPipe* aims to be an alternative mode to the writing of macros, our software employs various strategies to avoid the challenges encountered while preparing scripts. These include the memorization of the syntax, code organization, the additional instructions required for creating a batch processing pipeline, handling of error conditions, and the difficulty in accessing parameter documentation. The necessity to learn a syntax is already resolved by the visual programming concept. The only requirement is that users need to know the name and purpose of various *ImageJ* operations. These can be formed into pipelines via our GUI without explicit knowledge of program design. An issue with many VPLs is that with an increasing number of nodes, the pipeline becomes less understandable. *JIPipe* mitigates this issue by allowing custom documentations and comments. Additionally, we included a unique compartmentalization feature that separates a pipeline into visually distinct subunits.

While *JIPipe* offers a more accessible alternative to macro programming, our software must be approachable by users that are only familiar with the interactive *ImageJ* GUI. One challenge is introduced by the change in paradigm from an interactive workflow focused on concrete data representations to a more abstract environment that is built around processing operations. The second problem is that various *ImageJ* commands are based on GUI operations and thus cannot be perfectly encapsulated into *JIPipe* nodes. For example, operations on *ImageJ* ROIs rely on an interactive selection that cannot be trivially adapted to a batch processing workflow.

As the basic components of a pipeline are nodes, users must be able to find and recover the required operations within a set of over 1000 available functions, which is simplified by various GUI functions. Nodes are organized into a manually curated menu that is structured in a data-oriented way. For example, users will find image thresholding operations in a menu "Images > Threshold", while nodes that import or generate data are sorted into the "Add data" category. We complemented our manually curated menu by a search function that allows to find nodes by their name, description, and functional category, thus aiding researchers that are already familiar with operations from *ImageJ* or other software. A disadvantage of the menu and global search function is that they are not context-sensitive. Instead,

users are presented with the list of all nodes, which makes it difficult for new users to decide how to obtain specific input data or how to continue the processing of an output. To guide users with the creation of workflows, we included a feature termed "Algorithm finder" that targets a user-selected input or output and presents them with a sorted list of compatible nodes. These can be easily recovered by an additional a feature for saving a selection of nodes into a dedicated user-generated list. A single node is usually not sufficient to apply any relevant image processing, meaning that new users must familiarize themselves with the correct order of operations. We designed the *JIPipe* GUI to provide a clear labeling of the expected data types and role of inputs and outputs directly within the node. Our design thus enables the full visual understanding of a node's function without referring to the documentation. Finally, the project-based format implemented in our software allows the sharing of pipelines via publications and online communities. Similar to the macros, this allows users to learn the appropriate order of operations on examples.

A motivation behind the usage of *ImageJ* macros is their ability to create scalable batch processing pipelines. The process of generating such workflows involves additional code to capture multiple inputs, execute workloads per data set, and export results in a standardized format. To remove this overhead, we designed *JIPipe* to provide "zero-cost scalability", meaning that single-data projects can be designed to automatically scale to larger data sets without the need for changes in the set of nodes. To simplify the scalability of pipelines, *JIPipe* always manages tables of data and metadata, which is similar to the layout applied in *KNIME* [14]. These data tables allow the processing of multiple items per node by iterating the table row-wise, thus mitigating the additional complexity introduced by loop control structures. A disadvantage of table-based iteration is that it is only trivial for nodes that comprise one input, as the correct order of multiple input table rows cannot be ensured. A solution applied by *KNIME* is to only accept exactly one table that is iterated over its rows. The inputs for the underlying operation are extracted from a specific column that can be selected by the user, or automatically determined by a heuristic. *KNIME*'s approach has multiple disadvantages that include the reliance on existing data items to configure the inputs, the increased complexity of merging multiple data tables via a dedicated node, and restrictions on the display of multiple inputs in the GUI. The last point must be specifically highlighted, as a consistency between displayed and effective inputs reduces the necessity of reading documentation to understand the behavior of a node. To simplify the concept of table-based iteration, *JIPipe* accepts multiple data tables per node and joins corresponding data items internally. This is enabled by the enforcement of constraints: our table model introduces the concept of a "primary data column" that exists exactly once per table and stores data of a predefined *JIPipe* data type. Any additional column can either contain text metadata or generic metadata. Due to these restrictions, an automated algorithm is capable of iterating through multiple inputs by grouping rows via their text annotations. Due to the presence of a primary data column, it is clear which data should is passed the node-encapsulated operation. Another benefit of our approach is the intuitive relation between data iteration and metadata. For example, batch processing can

be implemented based on experimental conditions and biological sample identifiers that are already present in the raw data sets.

The final aspect of *ImageJ* macros is their use in achieving reproducibility, which is an important aspect of the FAIR principles and RDM. *JIPipe* goes beyond the capabilities of *ImageJ* by providing an advanced standardized project format that additionally captures important information about the generating environment, i.e., the version of *JIPipe* and all utilized plugins. Consequently, projects can be fully reproduced in their entirety by selecting the software versions specified in the pipeline metadata. Another unique feature of *JIPipe* is its standardized result storage format that saves data together with all related metadata and type information. Thus, *JIPipe* can both write and read data of this format, making it easy to further process existing results. Due to the standardization and its simple structure, third-party programs can easily implement our format to communicate with *JIPipe*, or utilize the storage specifications as FAIR-compatible mode of data exchange.

To summarize, our software tool *JIPipe* greatly contributes towards the ease of access to advanced and reproducible image analysis pipelines, thus acting as a bridge between experimentalists and computer scientists. This is achieved by its visual mode of programming, in combination with careful design choices that guide users through the variety of image analysis operations. Due to the automated RDM features implemented in *JIPipe*, users can trivially implement the FAIR principles.

Perspectives

Platform for teaching image analysis. *JIPipe* is based around a visual workflow that simplifies data management, parameter validation, and branching to test multiple analysis approaches. This makes our tool an ideal platform to teach image analysis methods. In the following paragraphs, we will briefly discuss the current features of our software that facilitate the setup of teaching environments and how these can be further improved.

To allow the highlight of a node's functionality, we included a feature that allows the custom naming of nodes and its inputs and outputs. The labeling function is complemented by a customizable description field that is capable of embedding images, links, and formatted text. To highlight specific areas in a pipeline, *JIPipe* supports the creation of comment nodes. These can be connected to any node for highlighting purposes and customized to a high degree to differentiate between different types of areas, for example, by changing the icon and background color.

A useful tool in the teaching of text programming languages is a focus on the program structure prior to the involvement of program code [43]. The concept of designing a program structure can be adapted to VPLs by providing students with tools to clearly separate the general structure of a visual program from the concrete implementation as sets of workloads. *JIPipe*, *KNIME*, *Icy*, and other VPLs provide nodes that contain a sub-workflow. While such "group nodes" can be utilized to outline the functionality of a pipeline, they are not clearly separated from the set of functional nodes. *JIPipe* already features a mode of separating

pipelines into structural units via the creation of "graph compartments". These manage the structure of a workflow via a dedicated hyper-pipeline that defines the characteristics of the available structural components, as well as how they exchange data. Our unique concept thus enables the purely structural modelling of an analysis without the involvement of concrete algorithms. For example, teachers can set up a project that consists of the common steps of image analysis [93]. The resulting structural information then acts as guidelines for students to simplify the choice of nodes.

JIPipe currently provides over 1000 different node types, which can overwhelm students who are not familiar with our tool. To simplify the development of pipelines for practical exercises, we included two features: the project overview page, and node templates. To communicate information about a project, including authors, contact information, citations, and instructions, *JIPipe* features a project-specific information GUI. The interface can be freely filled with a custom description, bookmarks to nodes of interest, and references to parameters. These features can be utilized to provide instructions, illustrative figures, and references to literature. To facilitate more advanced courses, where students are expected to select nodes from a curated selection, we introduced a feature that allows the manual addition of nodes and their parameters into a dedicated list. This function termed "node templates" can be used to predefine the set of nodes that are utilized within an exercise.

Expansion of 3D image analysis and visualization tools. *JIPipe* already includes features provided by *MorphoLibJ* [78] and *CLIJ* [55] that allow 3D object tracking, 3D noise filtering, and 3D morphological operations. Yet, there is currently a variety of missing functions, for example, 3D thresholding, 3D edge detection, or the extraction and management of 3D ROIs. The basis for including the missing functionality is the *3D ImageJ Suite* [102] library and its standardized interface *TAPAS* [49]. As we already successfully employed the automated encapsulation of functions of *ImageJ2* [115] and *CLIJ* [55], it is attainable that the algorithms provided by *TAPAS* can be easily included via a similar approach. An exception to this procedure is the integration of the 3D ROI feature that requires the implementation of a custom *JIPipe* data type.

Another weakness of *JIPipe* is the current lack of visualization for 3D data. Our custom image viewer can only display 2D images and thus splits 3D data into individual slices that make it difficult to see structures and evaluate results. To display 3D data appropriately, our viewer needs to be extended with 3D rendering, which is a non-trivial task due to the high resolution of bioimage data. A viewer must also support multiple rendering modes, including the presentation of pixels as volumetric cloud, surfaces, and slices. Due to the challenges involved in implementing such a software, it is more efficient to make use of already existing *Java*-based 3D visualization libraries. Examples include the *3D ImageJ Suite* [102], the 3D Viewer plugin [120], and the *SciView* platform [54]. An alternative is the integration of non-*Java* image viewers, for example, *Napari* [31], *Imaris* (Oxford Instruments), or *Arivis* (Arivis AG). The data transfer between *JIPipe* and non-*Java* software is facilitated by interfacing libraries, for example, *pyimagej* [46], or

the *imglib2-imaris-bridge* (Oxford Instruments). If no such API is available or developed, the only option left is to write image files to the hard drive prior to the visualization, which is a time-consuming process.

Integration of state-of-the-art object tracking algorithms. A common task in the analysis of time series is the tracking of organisms over time to quantify their behavior [93]. To a limited degree, this is already possible in the current version of *JIPipe* by utilizing a node designed for finding connected ROI components in three dimensions. As a 2D time series is equivalent to a 3D image, our ROI processing operation thus can be successfully applied. A disadvantage of our existing tracking algorithm is that it was not designed to handle the difficulties in tracking biological objects. There are two common issues that can occur in such analyses: first, the segmentation process is not always capable of fully separating multiple objects due to their proximity, thus producing clusters and an unwanted merge of multiple tracks; second, the segmentation can fail in specific frames, meaning that our simple algorithm loses track of the object.

Methods that are capable of resolving the listed issues are implemented in dedicated tracking software, for example, *AMIT* [3], the *OpenCV* [20] tracking module, or *TrackMate* [130]. Specifically *TrackMate* is suitable for the integration into *JIPipe*, as it is written in *Java* and already provides an interactive GUI for *ImageJ*. Due to the flexibility of *JIPipe*, it is capable of encapsulating the *TrackMate* features, including the setup of the tracking parameters, integration of alternative segmentation methods, visualization of tracks, detection of branches, and generation of statistics. For example, users can be provided with a tracking node for generating the track data in a standard format. These can be consumed by a set of other nodes to split and filter tracks, generate visualizations, calculate statistics, and convert the tracks into ROI.

Inclusion of advanced deconvolution operators. As established in section 1.3.3, it is beneficial for the analysis to suppress image degradation processes via deconvolution. Yet, *JIPipe* currently does not provide any node capable of applying such an operation. Users are instead forced to execute this preprocessing step outside *JIPipe*, for example, via *Huygens* (SVI), *Imaris* (Oxford Instruments), *DeconvolutionLab2* [116], or *Iterative Deconvolve 3D* [38]. While *Huygens* is a commercial tool and thus cannot be easily integrated into *JIPipe*, both *Iterative Deconvolve 3D* and *DeconvolutionLab2* are built on *ImageJ* and thus can be easily encapsulated into our software. A benefit of integrating multiple deconvolution methods into *JIPipe* is the simplified application of different deconvolution algorithms and parameter sets across multiple images. The results can be compared with the purpose of aiding researchers with the correct choice of parameters, for example, via the *DeconvTest* [92] framework.

Visual design of deep neural networks. *JIPipe* already provides limited capabilities to utilize deep learning techniques via its *Cellpose* [126] integration that will be expanded to encompass *Omnipose* [34]. Both *Cellpose* and *Omnipose* are

based on *Pytorch* [106], which simplifies the creation, training, and application of DNNs via an API. The disadvantage of *Pytorch* is that its functions are inaccessible to non-programmers. This is also the case for similar libraries, including *Tensorflow* [1], and *MXNet* [28].

An important feature of deep learning frameworks is the creation of custom architectures by defining and connecting the layers (see Figure 1.10b). As this process can be visually represented as flow chart, the pipeline described by a VPL can serve as basis for the creation of a network architecture. The visual design of DNNs is for example, implemented in the web-based learning platform *Tensorflow Playground* (Google LLC), and the commercial VPL *Deep Learning Studio* (Deep Cognition Inc.). *JIPipe* can be extended to provide an open source alternative to these commercial tools that allows the creation of viable deep learning models. The challenge of adapting *JIPipe* to the design of DNN architectures is that our software was designed for data processing, while the generation of a model is based on the structural information contained within the graph. Due to the flexible data structure implemented in our software, it is feasible to base the build-up of the network layers on a newly designed data type. Its purpose is to store a representation of the graph of layers that are iteratively added via nodes. A dedicated operation converts the model information into a DNN that can be utilized for training and prediction.

Due to the ease of access to the functionality of DNN libraries, the DNN design feature could also serve a purpose in teaching, as the effect of various architectures can be interactively explored.

Remote data exchange and processing. We designed *JIPipe* for the creation and execution of image analysis workflows on local computers, under the assumption that the hardware has sufficient resources to execute the operations. For example, we apply our custom algorithms on purpose-built workstation machines with multiple central processing unit (CPU) cores, a modern graphics card with 24 gigabytes of video random access memory (vRAM), and 256 gigabytes of random access memory (RAM). Such an environment is uncommon in standard office computers, meaning that only small data sets can be processed on many machines. A current trend is "cloud computing", which offloads the computationally expensive workloads to compute servers that are either hosted at the local institute, or by commercial services, for example, *AWS* (Amazon.com, Inc.), *Azure* (Microsoft Corporation), or *Google Cloud Platform* (Google LLC). It is not always possible to run *JIPipe* on a remote server via a virtual desktop, due to missing GUI libraries, administrator settings, or a slow internet connection.

A solution is the implementation of a dedicated server component that allows remote access to *JIPipe* functionality without the need for a graphical interface. This server provides a REST API that communicates data and pipelines via hypertext transfer protocol (HTTP). The protocol must be capable of managing server connections, reporting the status of remote operations, scheduling workflows on the server, exchanging settings, and securely communicating inputs and results between the client and host. Additionally, extensive changes to the existing cache and result display functions of *JIPipe* must be made to reduce wait times and the

network load. For example, it should be avoided to download all results in their entirety and only request the full data if absolutely needed. To support these operations, the REST API must be designed give access to specific rows in the output table of a node. As the data format implemented in *JIPipe* already allows the selection of data items, it is easy to adapt these principles to remote resources. For example, a local path `C:/data/auto-threshold/output/1/data.tif` can be trivially mapped into a uniform resource locator (URL) `https://localhost/data/auto-threshold/output/1/data.tif` that provides the selected data via HTTP.

The server component can be implemented as part of the *JIPipe* core API, which opens a use outside remote communication: due to the standardization, a local server can act as standardized interface to exchange data between *JIPipe* and external tools without the need for intermediate storage on the hard drive. For example, the *Python* image viewer *Napari* [31] can be expanded to offload image processing tasks to *JIPipe* by executing the appropriate REST API functions.

5.3 Visual programming pipelines quantify toxicity of fungus endosymbionts to nematodes

Due to the number of species, and the complex multi-layered structure of the environment, the interactions between soil-dwelling microorganisms are still largely unknown [45]. While fungi of the species *Mortierella verticillata* are known to play a role in the promotion of plant growth, the function of their endosymbiotic bacteria is to this day not entirety deciphered. In our publication [24], we aimed to characterize the relationship between *Mortierella verticillata*, its endosymbiotic bacteria, and the fungivorous nematode species *Aphelenchus avenae*.

We discovered via genomic assembly that one of the four identified endosymbiotic bacteria, *Candidatus Mycoavidus necroximicus*, produces cytotoxic macrolactones. To quantify the effectiveness of the toxin, we established viability assays with *Caenorhabditis elegans* that captured their movement. The quantification of this mobility allowed the identification of the inhibitory concentration at 50% (IC_{50}), thus proving the cytotoxic effect of the identified macrolactone. Further experiments with co-cultures of *Aphelenchus avenae* and *Mortierella verticillata* showed that the toxin is solely produced by *Candidatus Mycoavidus necroximicus* and inhibits the feeding behavior of the fungivorous nematode.

Our study was possible due to the precise quantification of the cytotoxic effects that were visible in an impact on the mobility of the nematodes. As a result of the high number of data sets, nematodes, and frames, the quantification of the captured transmitted light (TL) microscopy time series could not be applied manually. For example, to manually quantify the mobility of 100 nematodes in a movie with 80 frames, a researcher must segment 8000 objects, which is time consuming even without considering the necessary calculations to estimate the mobility of each organism. Instead, we developed a fully automated image analysis workflow that is capable of reproducibly quantifying the effect of toxins on hundreds of nematodes in parallel. As this operation was applied by a computer, it only required

a fraction of the time of an equivalent manual analysis. Initially the pipeline was developed as *ImageJ* macro, which resulted in hard-to-maintain code. For example, tracking individual nematodes across the time series was challenging, as *ImageJ* only supports one ROI list at the same time. Our macro thus resorted to a complicated protocol that involved the storage of ROI lists into temporary directories. To resolve these issues, we converted the set of image operations to our newly developed VPL *JIPipe* [47]. This resulted in a greatly simplified pipeline, due to the automated data management implemented in *JIPipe*.

The fully automated pipeline applies various image processing steps to enhance the characteristic visual features of nematodes, followed by a step for segmentation and morphological processing. The resulting 2D ROIs were tracked over time by a purpose-built *JIPipe* node. As we captured each individual nematode as series of *ImageJ* ROIs, the mobility could be easily calculated by dividing the total footprint by the nematode area. A comparatively low number thus indicated a dead or inhibited organism. Due to the number of experiments and individual nematodes, statistical methods could be utilized to find significant differences between various experimental conditions. At same time, the measurements allowed to estimate the IC_{50} of the toxin, thus highlighting the versatility of quantitative approaches.

Perspectives

Improved nematode tracking. The current nematode tracking implementation relies on the overlap of ROIs from different time points. While our approach was successful due to the high image quality, there were still events where multiple nematodes were falsely merged into one ROI. Our image analysis pipeline is based around intensity thresholding and thus does not incorporate the morphological characteristics of nematodes. While our custom tracking algorithm can detect merging events, these are resolved by the exclusion of the ROI. Consequently, the affected track was interrupted, which has an impact on the distribution of mobility ratios.

An alternative to our custom-built tracking algorithm is the utilization of an existing library, for example, *wrMTrck* [100] or *TrackMate* [130]. The benefit of the purpose-built tools is that they utilize multiple information sources to resolve clusters, wrong segmentation results, and other complex conditions. For example, *TrackMate* is capable of handling the fusion and splitting of objects. Additionally, tracking tools will generate detailed statistics that, for example, capture morphological changes over time.

The success of the tracking operator can be additionally enhanced by an improved segmentation pipeline. Due to the directional lighting, all nematodes in the captured TL images show a visible line close to their medial axis. By enhancing and extracting this shape, clusters can be resolved by the application of a Voronoi partitioning, i.e. dividing the image into spaces that are defined by the medial axes. As each partition contains only pixels of the same nematode, the erroneous segmentation can be improved by splitting segmented regions according to the partition borders. This improved segmentation then can be tracked via our custom-built algorithm, or supplied to a more advanced object tracker.

Extraction of advanced shape features. Currently, our image analysis pipeline generates a mobility measurement that allowed us to find a significant impact of the toxin level. This is achieved by comparing the area of each nematode with its total footprint. Yet, as each object is fully tracked over the whole time series, more detailed measurements can be extracted from the data. The mobility ratio is a scalar measurement, as it condenses the information into one number. It is also possible to capture multiple points into a "feature vector" or even "feature matrix" that contain more data about the measured subject, and thus can be of use for machine learning or modelling. For example, the K-means algorithm can be applied to classify nematodes into multiple groups, which can give further insight into their behavior. Another use of advanced measurements is in computer models that can be calibrated to mirror the patterns described in the features. In the following paragraphs, we will briefly describe various alternative measurements for the analysis of nematode behavior.

The mobility of nematodes relies on the generation of the curving motions [105]. Consequently, it is of interest to measure the mean curvature of each tracked organism and how it changes over time. To calculate the average curvature, an algorithm first detects the local curvature around each contour point by fitting a circle [39], followed by an averaging of the resulting values. A feature vector then can be obtained by collecting the average curvatures for each time point. By measuring the curvature of multiple key points, a feature matrix can be generated. Nematodes have a distinct polar shape, thus making the anterior and posterior ends suitable starting locations for sampling stable contour points from each ROI. For example, the lines between the anterior and posterior points can be split into segments of equal length that serve as basis for multiple key points.

An alternative to the direct measurement of curvature is the description of the nematode outline via weighted standard shapes. One approach is to utilize Fourier-based shape analysis methods [56] that decompose 2D shapes into sine and cosine curves. Their coefficients then can be collected to form a feature matrix. The benefit of sine and cosine features is in their use for computer simulations, where the weights and frequencies can be replicated in computer models. Another benefit of Fourier-based shape analysis is that high-frequency noise can be removed, thus reducing the complexity of the feature matrix. Alternatively, there are purpose-built approaches for the extraction of compact features from nematodes [53].

Specificity of the cytotoxic agent. We successfully identified that *Candidatus Mycoavidus necroximicus* produces macrolactones that are toxic against *Aphelenchus avenae* and *Caenorhabditis elegans*. This raises the question on the specificity of the toxin, i.e., if it also targets bacteria, amoebae, or other fungi. The existing approach based on image-based quantification can be adapted to cultures with bacteria and fungi. While bacteria and amoebae are mobile, allowing to re-use our existing pipeline, fungi are usually immobile. An alternative measurement can be extracted by testing the toxicity to fungal spores by tracking their germination behavior and comparing their areas between the first and last frames.

Alternative to fertilizers and pesticides. The current standard approach for improving the soil quality is by the utilization of fertilizers. These have a multitude of side effects on the ecosystem [10], for example, the accumulation of toxic heavy metals in groundwater and rivers [119]. Additionally, fertilizers play a role in the eutrophication of bodies of water, which is characterized by increased formation of algae, reduction in oxygen levels, and the subsequent death of fish and other water organisms. Another type of contamination are pesticides [2] that increase productivity of crops, but contaminate soil and water, damage plants, and kill beneficial microorganisms, insects, as well as birds. These disadvantages highlight the need for alternatives to fertilizers and pesticides that avoid unwanted side effects.

The fungus *Mortierella verticillata* that was the focus of our study already displays characteristics of both fertilizers and pesticides. As soil-dwelling fungi play a role in the extraction of nutrients from detritus, and removal of harmful agents from soil, they contribute towards the improvement of the soil quality and consequently to the yield of crops. At the same time, our study characterizes how *Mortierella verticillata* defends against fungivorous nematodes. As the detected cytotoxic agent also kills other nematode species, the toxin shares characteristics with pesticides that target plant-pathogenic nematodes.

While it is clear that our publication does not allow the development of a concrete strategy to replace fertilizers and pesticides, it contributes towards the understanding of complex soil ecosystems. To facilitate the generation of knowledge and new hypotheses, the known soil interactions can be integrated into computer models that allow the inference of hidden parameters, and rapid simulation of multiple experimental conditions *in silico*. Today, computer hardware has advanced to a degree that agent-based models (ABMs) [19] with millions of agents and complex interactions can be simulated in a reasonable time [29]. A soil model can be implemented as ABM that represents microorganisms as spherical shapes [17] within a non-discrete space. To represent elongated structures that represent worms or fungal hyphae, multiple spheres can be linked into a chain [79]. Advanced algorithms for 3D collision detection [142] alternatively allow the replacement of simple shapes by realistic 3D models that can be generated with computer programs, or from microscope data. To simplify the development of a computer model, existing purpose-built simulation frameworks can be utilized, for example, *Netlogo* [131], *GAMA* [127], or *MASON* [86].

5.4 Standardizing high-performance image analysis software enables the rapid quantification of big data volumes

In the previously discussed studies [59, 47, 24], we utilized the *ImageJ* platform to manage and visualize images, apply spatial filters, and generate measurements. The software is developed in *Java*, which is a programming language focused on portability. This is achieved by compiling code into an intermediate language that

can only be executed by a virtual machine (VM) [84]. By porting the VM to multiple computer platforms, *Java* programs thus can be executed on different operating systems without the need for recompilation. The drawback is that virtualization limits the performance of operations [95], thus slowing the performance of image processing operations based on *Java*.

One way of optimizing the performance of a *Java*-based image analysis pipeline is the re-implementation into *C++* [63], which is a programming language that is compiled into machine instructions for a specific operating system. As the resulting program is already in a machine-readable form, a VM is made obsolete, thus removing its impact on the performance. Additional speedup is gained by the ability of *C++* compilers to optimize the machine code [50], for example, by removing spurious calculation steps, or by introducing CPU instructions that process multiple steps at once. Advanced compiler features for optimization purposes are already utilized by existing image processing libraries, including *OpenCV* [20], and *VTK* [123].

A major disadvantage of converting a *Java* software into a *C++* application is the required development time. For example, developers must find equivalent *C++* algorithms, add functions for management of data and parameters, and implement a CLI. Due to these challenges, we will briefly discuss alternative options. Generally, existing code can be optimized or replaced by algorithms with more efficient implementations. An example is the "Fast Hough Transform" [81] that greatly improves on the speed of an algorithm to find line-like structures inside images. If code cannot be optimized, performance-critical algorithms can be outsourced into compiled libraries via the *Java Native Interface* (JNI) [82] (*Java*), *ctypes* (*Python*), or *Rcpp* [40] (R). Alternatively, performance-critical segments can be automatically converted into compiled code as implemented in *Cython* [12] and *Numba* [76]. If the existing solutions for integrating performance-critical sections into *Java* code are insufficient, a re-implementation into *C++* cannot be avoided. Our publication [48] introduces a software framework termed *MISA++* that greatly simplifies the development of high-performance image quantification pipelines via *C++*. This is achieved by providing reusable components that standardize data and metadata management, parallelization, parameters, and user interaction. By utilizing these functions, developers can focus on the implementation of their image analysis workflow, thus reducing the required development time and complexity of the source code.

Due to the high standardization of the *MISA++* framework, our software is capable of generating human- and machine-readable descriptions of their underlying workflow. For example, users can query via a CLI the expected inputs, algorithm parameters with default values and documentation, as well as the structure of the results. As this information is provided in the JSON format and thus can be parsed by a computer, software based on *MISA++* can be easily integrated into automated analysis scripts. The high standardization thus facilitates the implementation and propagation of the FAIR principles [140]: first, as *MISA++* automatically enforces that any software based on our framework adheres to strict and open standards for parameter and data storage; second, algorithms are standardized and modular to facilitate their re-use in other projects.

An alternative use of our standardized description protocol is in the automated generation of a GUI that encapsulates the features of the *C++* software into a user-friendly environment. In our study, we presented an implementation of this feature in form of a plugin for *ImageJ* that is capable of providing a GUI for any *MISA++*–based software.

To summarize, our *MISA++* framework simplifies the development of advanced, high-performance, and FAIR-compatible image analysis pipelines, while providing a user-friendly and standardized GUI around *C++* applications.

Perspectives

Utilization of graphics hardware. While *MISA++* is mainly designed for the parallelization of CPU workloads, there is an ongoing trend in the development of image processing algorithms that utilize graphics processing units (GPUs) [55, 42]. This is motivated by the highly parallel nature of GPUs that can significantly improve the speed of various algorithms: for example, a GPU-accelerated circular Hough transform can be up to 64 times faster than a CPU-based implementation [65].

OpenCV and *VTK* can already be configured to utilize a GPU. As *MISA++* integrates *OpenCV*, users of our framework only need to apply minor configurations to process images via the GPU.

To allow the development of custom GPU algorithms, developers can target the *C* [64] and *C++* [63] APIs of *CUDA* [101] (Nvidia Corporation) and *OpenCL* [60]. Directly utilizing *CUDA* and *OpenCL* in program code is challenging, due to the necessity of manual memory management and compilation of GPU-specific code via dedicated software. *OpenAcc* [103] resolves these disadvantages by extending the *C++* language with a feature that allows the definition of performance-critical sections. Similar to *Cython* [12] and *Numba* [76], these regions are automatically replaced by accelerated code. At the same time, *OpenAcc* fully automatically handles the setup of computation devices and memory transfer. Performance-critical sections are marked via a `#pragma` statement, meaning that code designed for *OpenAcc* is always valid *C++* code. Consequently, any *OpenAcc*-based project can be built without a special compiler.

MISA++ thus does not need specific functions for the handling of GPU processing, as such implementations are either provided by third-party image processing libraries, or can be easily included by any developer via *OpenAcc*.

Mixed-resolution analysis via direct communication with a microscope. The capture of images via confocal- or light sheet fluorescence microscopy (LSFM) involves a balancing act between the resolution and photo damage: while low-resolution images can be generated quickly and only produce minor photobleaching, the details might be not sufficient to capture all processes of the experiment. These are only contained in high-resolution images that are generated slowly and thus limited due to the high damage. A mixed-resolution approach combines the benefits of both high- and low-resolution imaging: the sample is first captured in

a low resolution with the purpose of identifying ROIs. The following generation of high-resolution data is then restricted to the ROIs. A major disadvantage of mixed-resolution microscopy is that it currently relies on manual user inputs to determine the ROIs, which is hard to reproduce, time consuming, and difficult to achieve for 3D data.

Our *MISA++* framework allows the development of image analysis pipelines that automatically segment and quantify image data. Our tool can be combined with the open source *C++* library *µManager* [41] that can instruct the microscope to capture a specific area. This allows the implementation of a fully automated mix-resolution microscopy approach that is entirely controlled by a *MISA++*–based software. This tool would instruct *µManager* to first produce a low-resolution image of the whole specimen that is automatically segmented by an image analysis pipeline. The produced ROIs are used as basis for the capture of the high-resolution images that are automatically processed and quantified.

Integration of *MISA++* into *JIPipe*. *MISA++* standardizes the communication of the parameters and data, which allowed us to develop an *ImageJ*-based GUI for executing any software developed with the framework. Another feature of the plugin is a VPL that allows non-programmers to form pipelines consisting of *MISA++* software modules. As *JIPipe* can be easily extended with plugins to integrate the functionality of non-*Java* software, it is viable to embed the features of the *MISA++* plugin into *JIPipe*, thus unifying the integration of *MISA++* software within *ImageJ*.

One point that needs to be considered before developing a *MISA++* extension for *JIPipe* is the modernization of the data and metadata storage implementation that is already present in the *MISA++* framework. *MISA++* currently stores data items in a hierarchy of directories that is based around a unique data set identifier and the algorithm that generated the result. A weakness of the current *MISA++* data model is that the location within the file system hierarchy is essential to access information about a data item, i.e. its type and associated metadata. The consequence is that results produced by *MISA++* cannot be separated from its output folder. The *JIPipe* data model organizes data into tables where each row represents one data item. The columns can contain text annotations or any other *JIPipe*-compatible data. Each data table is stored within its own standardized directory that saves all characteristics in a JSON metadata file, meaning that the directory can be freely moved without losing information. Replacing the *MISA++* data format with a modernized implementation based on *JIPipe* would thus make *MISA++* more versatile. A data exchange between our *Java* tool and *C++* can then be trivially implemented via temporary file-based storage, or could utilize our proposed web-based data exchange API.

Another alternative mode of data exchange is to compile *MISA++* tools into software libraries to be integrated into *Java* via the JNI [82], thus establishing a direct communication without the involvement of protocols. One issue with a JNI-based integration is that *MISA++* is not designed for communicating via a *C* [64] API. For example, there is no function that queries a software library for all available *MISA++* workflows and their characteristics. Adding this feature to *MISA++*

opens a possibility to further simplify the development and deployment of *C++* software: currently, developers must write *C++* code to generate a standardized CLI for their *MISA++* project. The consequence is that even simple *MISA++*–based tools consist of multiple code projects. By moving these features into a dedicated *C* and *C++* APIs, the per-project CLI is obsolete. Instead, their functionality can be condensed into a single independent application responsible for managing any library built on *MISA++*.

5.5 Concluding remarks

This thesis comprises the development and application of modern software frameworks that allow the fully automated characterization of host-pathogen interactions via the quantitative analysis of microscopy or tomography images. In particular, the goal of this thesis is to contribute towards the propagation of the FAIR principles [140] in the field of IbSB by the development of standardized, reproducible, high-performance, and accessible software for the quantification of interactions in biological systems.

These goals were successfully realized by our reproducible algorithm *Mcat* for the analysis of MSOT data, the introduction of the software *JIPipe* that opens the development of advanced and FAIR-compatible *ImageJ* pipelines to non-programmers, its application in the characterization of a three-species interaction by an interdisciplinary team, and the debut of the highly standardized and FAIR-compliant *MISA++* framework capable of processing big data volumes. These developments are ideal starting points for future collaborations with life scientists and medical doctors to develop improved experiments, imaging methods, advanced algorithms, and computer models in the spirit of the image-based systems biology approach.

Bibliography

[1] Martín Abadi et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467* (2016).

[2] Md Wasim Aktar, Dwaipayan Sengupta, and Ashim Chowdhury. "Impact of pesticides use in agriculture: their benefits and hazards". In: *Interdisciplinary toxicology* 2.1 (2009), p. 1.

[3] Naim Al-Zaben et al. "Automated tracking of label-free cells with enhanced recognition of whole tracks". In: *Scientific reports* 9.1 (2019), 1–10.

[4] Javier Antonio Alfaro et al. "The emerging landscape of single-molecule protein sequencing technologies". In: *Nature methods* 18.6 (2021), pp. 604–617.

[5] Chris Allan et al. "OMERO: flexible, model-driven data management for experimental biology". In: *Nature methods* 9.3 (2012), 245–253.

[6] Ignacio Arganda-Carreras et al. "Consistent and elastic registration of histological sections using vector-spline regularization". In: *International Workshop on Computer Vision Approaches to Medical Image Analysis*. Springer. 2006, 85–95.

[7] Aristotle. *Topics*. English. Translated from ancient Greek by W. A. Pickard-Cambridge. Accessed on May 16th, 2022. The Internet Classics Archive, 350 BCE. URL: `http://classics.mit.edu/Aristotle/topics.html`.

[8] Judith M Bain et al. "Non-lytic expulsion/exocytosis of Candida albicans from macrophages". In: *Fungal genetics and biology* 49.9 (2012), 677–678.

[9] Guido Barbaglia, Simone Murzilli, and Stefano Cudini. "Definition of REST web services with JSON schema". In: *Software: Practice and Experience* 47.6 (2017), pp. 907–920.

[10] Riccardo Basosi et al. "Mineral nitrogen fertilizers: environmental impact of production and use". In: *Fertil. Compon. Uses Agric. Environ. Impacts, Nova science publishers. Lopez-Valdez, F and Fernandez-Luquenos, F, New York* (2014), 3–44.

[11] M Bauer et al. "Sepsis". In: *Der Anaesthesist* 55.8 (2006), 835–845.

[12] Stefan Behnel et al. "Cython: The best of both worlds". In: *Computing in Science & Engineering* 13.2 (2011), 31–39.

[13] Ilya Belevich et al. "Microscopy image browser: a platform for segmentation and analysis of multidimensional datasets". In: *PLoS biology* 14.1 (2016), e1002340.

[14] Michael R Berthold et al. "KNIME-the Konstanz information miner: version 2.0 and beyond". In: *AcM SIGKDD explorations Newsletter* 11.1 (2009), 26–31.

[15] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006. ISBN: 978-0387-31073-2.

[16] Marco Blickensdorf, Sandra Timme, and Marc Thilo Figge. "Comparative assessment of aspergillosis by virtual infection modeling in murine and human lung". In: *Frontiers in immunology* 10 (2019), p. 142.

[17] Marco Blickensdorf, Sandra Timme, and Marc Thilo Figge. "Hybrid agent-based modeling of Aspergillus fumigatus infection to quantitatively investigate the role of pores of Kohn in human alveoli". In: *Frontiers in microbiology* (2020), p. 1951.

[18] Kevin M Boergens et al. "webKnossos: efficient online 3D data annotation for connectomics". In: *nature methods* 14.7 (2017), pp. 691–694.

[19] Eric Bonabeau. "Agent-based modeling: Methods and techniques for simulating human systems". In: *Proceedings of the national academy of sciences* 99.suppl 3 (2002), 7280–7287.

[20] Gary Bradski and Adrian Kaehler. "OpenCV". In: *Dr. Dobb's journal of software tools* 3 (2000).

[21] Frank J Bruggeman and Hans V Westerhoff. "The nature of systems biology". In: *TRENDS in Microbiology* 15.1 (2007), pp. 45–50.

[22] Andreas Buehler et al. "High resolution tumor targeting in living mice by means of multispectral optoacoustic tomography". In: *EJNMMI research* 2.1 (2012), 1–6.

[23] P Burra and A Masier. "Dynamic tests to study liver function". In: *European review for medical and pharmacological sciences* 8 (2004), 19–22.

[24] Hannah Büttner et al. "Bacterial endosymbionts protect beneficial soil fungus from nematode attack". In: *Proceedings of the National Academy of Sciences* 118.37 (2021).

[25] Arturo Casadevall and Liise-anne Pirofski. "Host-pathogen interactions: basic concepts of microbial commensalism, colonization, infection, and disease". In: *Infection and immunity* 68.12 (2000), 6511–6518.

[26] Arturo Casadevall and Liise-anne Pirofski. "Host-pathogen interactions: redefining the basic concepts of virulence and pathogenicity". In: *Infection and immunity* 67.8 (1999), 3703–3713.

[27] Eugene Charniak. *Introduction to Deep Learning*. The MIT Press, 2018. ISBN: 978-0-262-03951-2.

[28] Tianqi Chen et al. "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems". In: *arXiv preprint arXiv:1512.01274* (2015).

[29] Wenan Chen et al. "Agent based modeling of blood coagulation system: implementation using a GPU based high speed framework". In: *2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE. 2011, 145–148.

[30] Richard Coico. *Immunology: a short course*. Seventh edition. John Wiley & Sons, 2015.

[31] napari contributors. *napari: a multi-dimensional image viewer for python*. DOI: 10.5281/zenodo.3555620.

[32] Zoltan Cseresnyes, Kaswara Kraibooj, and Marc Thilo Figge. "Hessian-based quantitative image analysis of host-pathogen confrontation assays". In: *Cytometry Part A* 93.3 (2018), 346–356.

[33] Zoltan Cseresnyes et al. "Quantitative impact of cell membrane fluorescence labeling on phagocytosis measurements in confrontation assays". In: *Frontiers in Microbiology* 11 (2020), p. 1193.

[34] Kevin J Cutler et al. "Omnipose: a high-precision morphology-independent solution for bacterial cell segmentation". In: *bioRxiv* (2021).

[35] Fabrice De Chaumont et al. "Icy: an open bioimage informatics platform for extended reproducible research". In: *Nature methods* 9.7 (2012), 690–696.

[36] Sophie Dennhardt et al. "Modeling hemolytic-uremic syndrome: in-depth characterization of distinct murine models reflecting different features of human disease". In: *Frontiers in immunology* 9 (2018), p. 1459.

[37] Django Software Foundation. *Django*. Version 2.2. May 5, 2019. URL: https://djangoproject.com.

[38] Robert Dougherty. "Extensions of DAMAS and benefits and limitations of deconvolution in beamforming". In: *11th AIAA/CEAS aeroacoustics conference*. 2005, p. 2961.

[39] Meghan Driscoll et al. "Cell Shape Dynamics: From Waves to Migration". In: *Bulletin of the American Physical Society* 56 (2011).

[40] Dirk Eddelbuettel et al. "Rcpp: Seamless R and C++ integration". In: *Journal of statistical software* 40.8 (2011), 1–18.

[41] Arthur D Edelstein et al. "Advanced methods of microscope control using μManager software". In: *Journal of biological methods* 1.2 (2014).

[42] Anders Eklund et al. "Medical image processing on the GPU–Past, present and future". In: *Medical image analysis* 17.8 (2013), pp. 1073–1094.

[43] Gregor Engels et al. "Teaching UML is teaching software engineering is teaching abstraction". In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2005, 306–319.

[44] Romain Fernandez and Cédric Moisy. "Fijiyama: a registration tool for 3D multimodal time-lapse imaging". In: *Bioinformatics* 37.10 (2021), 1482–1484.

[45] Noah Fierer. "Embracing the unknown: disentangling the complexities of the soil microbiome". In: *Nature Reviews Microbiology* 15.10 (2017), pp. 579–590.

[46] Niklas A Gahm et al. "New Extensibility and Scripting Tools in the ImageJ Ecosystem". In: *Current Protocols* 1.8 (2021), e204.

[47] Ruman Gerst, Zoltan Cseresnyes, and Marc Thilo Figge. "JIPipe: Visual batch processing for ImageJ". In: *ResearchSquare* (2022). Preprint. DOI: 10.21203/rs.3.rs-1641739/v1.

[48] Ruman Gerst, Anna Medyukhina, and Marc Thilo Figge. "MISA++: A standardized interface for automated bioimage analysis". In: *SoftwareX* 11 (2020), p. 100405.

[49] JF Gilles and T Boudier. "TAPAS: Towards Automated Processing and Analysis of multi-dimensional bioimage data [version 2; peer review: 2 approved]". In: *F1000Research* 9.1278 (2021). DOI: 10.12688/f1000research.26977.2.

[50] Matt Godbolt. "Optimizations in C++ compilers". In: *Communications of the ACM* 63.2 (2020), 41–49.

[51] Rafael C Gonzalez and Richard E Woods. *Digital image processing*. Pearson/Prentice Hall, 2008.

[52] Miguel Grinberg. *Flask web development: developing web applications with python*. " O'Reilly Media, Inc.", 2018.

[53] Bertalan Gyenes and André EX Brown. "Deriving shape-based features for C. elegans locomotion using dimensionality reduction methods". In: *Frontiers in behavioral neuroscience* 10 (2016), p. 159.

[54] Ulrik Günther and Kyle IS Harrington. "Tales from the Trenches: Developing sciview, a new 3D viewer for the ImageJ community". In: *arXiv preprint arXiv:2004.11897* (2020).

[55] Robert Haase et al. "CLIJ: GPU-accelerated image processing for everyone". In: *Nature methods* 17.1 (2020), 5–6.

[56] A John Haines and James S Crampton. "Improvements to the method of Fourier shape analysis as applied in morphometric studies". In: *Palaeontology* 43.4 (2000), 765–783.

[57] Thorsten Heinekamp et al. "Aspergillus fumigatus melanins: interference with the host endocytosis pathway and impact on virulence". In: *Frontiers in microbiology* 3 (2013), p. 440.

[58] Stefan W Hell and Jan Wichmann. "Breaking the diffraction resolution limit by stimulated emission: stimulated-emission-depletion fluorescence microscopy". In: *Optics letters* 19.11 (1994), pp. 780–782.

[59] Bianca Hoffmann et al. “Spatial quantification of clinical biomarker pharmacokinetics through deep learning-based segmentation and signal-oriented analysis of MSOT data”. In: *Photoacoustics* (2022), p. 100361.

[60] Lee Howes and Aaftab Munshi. “The OpenCL specification, version 2.0”. In: *Khronos Group* (2015).

[61] Liang-Kai Huang and Mao-Jiun J Wang. “Image thresholding by minimizing the measures of fuzziness”. In: *Pattern recognition* 28.1 (1995), 41–51.

[62] Kerstin Hünniger et al. “A virtual infection model quantifies innate effector mechanisms and Candida albicans immune escape in human blood”. In: *PLoS computational biology* 10.2 (2014), e1003479.

[63] ISO/IEC JTC 1/SC 22. *ISO/IEC 14882:2020 Programming languages — C++*. 6th ed. International Organization for Standardization, International Electrotechnical Commission, 2020.

[64] ISO/IEC JTC 1/SC 22. *ISO/IEC 9899:2018 Information technology — Programming languages — C*. 4th ed. International Organization for Standardization, International Electrotechnical Commission, 2018.

[65] Yasuaki Ito, Kouhei Ogawa, and Koji Nakano. “Fast ellipse detection algorithm using Hough transform on the GPU”. In: *2011 Second International Conference on Networking and Computing*. IEEE. 2011, 313–319.

[66] Vahid Jalili et al. “The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2020 update”. In: *Nucleic acids research* 48.W1 (2020), W395–W402.

[67] Josh Juneau et al. *The definitive guide to Jython: Python for the Java platform*. Apress, 2010.

[68] R Kapoor, S Ladak, and V Gomase. “MALDI-TOF based metabolomic approach”. In: *Int J Genet* 1 (2009), pp. 44–46.

[69] Jagat Narain Kapur, Prasanna K Sahoo, and Andrew KC Wong. “A new method for gray-level picture thresholding using the entropy of the histogram”. In: *Computer vision, graphics, and image processing* 29.3 (1985), 273–285.

[70] Srdjan Kesić. “Systems biology, emergence and antireductionism”. In: *Saudi journal of biological sciences* 23.5 (2016), pp. 584–591.

[71] Hiroaki Kitano. “Computational systems biology”. In: *Nature* 420.6912 (2002), 206–210.

[72] Hiroaki Kitano. “Perspectives on systems biology”. In: *New Generation Computing* 18.3 (2000), 199–216.

[73] Anika Klingberg et al. “Fully automated evaluation of total glomerular number and capillary tuft size in nephritic kidneys using lightsheet microscopy”. In: *Journal of the American Society of Nephrology* 28.2 (2017), 452–459.

[74] Kaswara Kraibooj et al. “Automated quantification of the phagocytosis of Aspergillus fumigatus conidia by a novel image analysis algorithm”. In: *Frontiers in microbiology* 6 (2015), p. 549.

[75] Mohammad Amin Kuhail et al. “Characterizing Visual Programming Approaches for End-User Developers: A Systematic Review”. In: *IEEE Access* (2021).

[76] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. “Numba: A llvm-based python jit compiler”. In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. 2015, 1–6.

[77] F. William Lawvere. “Metric spaces, generalized logic, and closed categories”. In: *Reprints in Theory and Applications of Categories* 1 (2002), 1–37.

[78] David Legland, Ignacio Arganda-Carreras, and Philippe Andrey. “MorphoLibJ: integrated library and plugins for mathematical morphology with ImageJ”. In: *Bioinformatics* 32.22 (2016), 3532–3534.

[79] Maolin Lei et al. “Real-Time Kinematics-Based Self-Collision Avoidance Algorithm for Dual-Arm Robots”. In: *Applied Sciences* 10.17 (2020), p. 5893.

[80] Mitchell P Levesque and Philip N Benfey. “Systems biology”. In: *Current Biology* 14.5 (2004), R179–R180.

[81] Hungwen Li, Mark A Lavin, and Ronald J Le Master. “Fast Hough transform: A hierarchical approach”. In: *Computer Vision, Graphics, and Image Processing* 36.2-3 (1986), 139–161.

[82] Sheng Liang. *The Java Native Interface: Programmer’s Guide and Specification*. Addison-Wesley Longman, Amsterdam, 1999. ISBN: 978-0201325775.

[83] Ping-Sung Liao, Tse-Sheng Chen, Pau-Choo Chung, et al. “A fast algorithm for multilevel thresholding”. In: *J. Inf. Sci. Eng.* 17.5 (2001), 713–727.

[84] Tim Lindholm et al. *The Java® Virtual Machine Specification*. Java SE 8 Edition. Feb. 2013. URL: `https://docs.oracle.com/javase/specs/jvms/se8/jvms8.pdf`.

[85] David G Lowe. “Distinctive image features from scale-invariant keypoints”. In: *International journal of computer vision* 60.2 (2004), 91–110.

[86] Sean Luke et al. “Mason: A multiagent simulation environment”. In: *Simulation* 81.7 (2005), pp. 517–527.

[87] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[88] John Maloney et al. “The scratch programming language and environment”. In: *ACM Transactions on Computing Education (TOCE)* 10.4 (2010), 1–15.

[89] Estibaliz Gómez-de Mariscal et al. “DeepImageJ: A user-friendly environment to run deep learning models in ImageJ”. In: *Nature Methods* 18.10 (2021), 1192–1195.

[90] Gabriel G Martins et al. “Highlights from the 2016-2020 NEUBIAS training schools for Bioimage Analysts: a success story and key asset for analysts and life scientists”. In: *F1000Research* 10 (2021).

[91] Claire McQuin et al. “CellProfiler 3.0: Next-generation image processing for biology”. In: *PLoS biology* 16.7 (2018), e2005970.

[92] Anna Medyukhina and Marc Thilo Figge. “DeconvTest: Simulation framework for quantifying errors and selecting optimal parameters of image deconvolution”. In: *Journal of Biophotonics* 13.4 (2020), e201960079.

[93] Anna Medyukhina et al. “Image-based systems biology of infection”. In: *Cytometry Part A* 87.6 (2015), 462–470.

[94] Mohammad H Mirhakkak et al. “Metabolic modeling predicts specific gut bacteria as key determinants for Candida albicans colonization levels”. In: *The ISME journal* 15.5 (2021), pp. 1257–1270.

[95] Jose E Moreira et al. “Java programming for high-performance numerical computing”. In: *IBM Systems Journal* 39.1 (2000), 21–56.

[96] Fionn Murtagh and Pedro Contreras. “Algorithms for hierarchical clustering: an overview”. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 2.1 (2012), 86–97.

[97] Juliette Mérian et al. “Fluorescent nanoprobes dedicated to in vivo imaging: from preclinical validations to clinical translation”. In: *Molecules* 17.5 (2012), 5564–5591.

[98] Pauline C Ng and Ewen F Kirkness. “Whole genome sequencing”. In: *Genetic variation* (2010), pp. 215–226.

[99] William S Noble. “What is a support vector machine?” In: *Nature biotechnology* 24.12 (2006), 1565–1567.

[100] Carmen I Nussbaum-Krammer et al. “Investigating the spreading and toxicity of prion-like proteins using the metazoan model organism C. elegans”. In: *JoVE (Journal of Visualized Experiments)* 95 (2015), e52321.

[101] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. *CUDA, release: 10.2.89*. 2020. URL: `https://developer.nvidia.com/cuda-toolkit`.

[102] Jean Ollion et al. “TANGO: a generic tool for high-throughput 3D image analysis for studying nuclear organization”. In: *Bioinformatics* 29.14 (2013), 1840–1841.

[103] OpenACC-Standard.org. *The OpenACCR Application Programming Interface*. Version 3.1. 2020.

[104] Wei Ouyang et al. “ImJoy: an open-source computational platform for the deep learning era”. In: *Nature methods* 16.12 (2019), pp. 1199–1200.

[105] Venkat Padmanabhan et al. “Locomotion of C. elegans: a piecewise-harmonic curvature representation of nematode behavior”. In: *PloS one* 7.7 (2012), e40121.

[106] Adam Paszke et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32 (2019), 8026–8037.

[107] Martinha Piteira and Carlos Costa. "Learning computer programming: study of difficulties in learning programming". In: *Proceedings of the 2013 International Conference on Information Systems and Design of Communication.* 2013, 75–80.

[108] Stephen M Pizer et al. "Adaptive histogram equalization and its variations". In: *Computer vision, graphics, and image processing* 39.3 (1987), 355–368.

[109] Johannes Pollmächer and Marc Thilo Figge. "Deciphering chemokine properties by a hybrid agent-based model of Aspergillus fumigatus infection in human alveoli". In: *Frontiers in microbiology* 6 (2015), p. 503.

[110] Johannes Pollmächer and Marc Thilo Figge. "Agent-based model of human alveoli predicts chemotactic signaling by epithelial cells during early Aspergillus fumigatus infection". In: *PloS one* 9.10 (2014), e111630.

[111] Rory M Power and Jan Huisken. "A guide to light-sheet fluorescence microscopy for multiscale imaging". In: *Nature methods* 14.4 (2017), pp. 360–373.

[112] Christophe J Queval, Roland Brosch, and Roxane Simeone. "The macrophage: a disputed fortress in the battle against Mycobacterium tuberculosis". In: *Frontiers in microbiology* 8 (2017), p. 2284.

[113] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention.* Springer. 2015, 234–241.

[114] Amir Rosenthal, Vasilis Ntziachristos, and Daniel Razansky. "Acoustic inversion in optoacoustic tomography: A review". In: *Current Medical Imaging* 9.4 (2013), 318–336.

[115] Curtis T Rueden et al. "ImageJ2: ImageJ for the next generation of scientific image data". In: *BMC bioinformatics* 18.1 (2017), 1–26.

[116] Daniel Sage et al. "DeconvolutionLab2: An open-source software for deconvolution microscopy". In: *Methods* 115 (2017), 28–41.

[117] Daniel Sage et al. "Quantitative evaluation of software packages for single-molecule localization microscopy". In: *Nature methods* 12.8 (2015), pp. 717–724.

[118] Jeremy Sanderson. *Understanding light microscopy.* John Wiley & Sons, 2019.

[119] Serpil Savci. "Investigation of effect of chemical fertilizers on environment". In: *Apcbee Procedia* 1 (2012), 287–292.

[120] Benjamin Schmid et al. "A high-level 3D visualization API for Java and ImageJ". In: *BMC bioinformatics* 11.1 (2010), 1–7.

[121] Uwe Schmidt et al. "Cell Detection with Star-Convex Polygons". In: *Medical Image Computing and Computer Assisted Intervention - MICCAI 2018 - 21st International Conference, Granada, Spain, September 16-20, 2018, Proceedings, Part II*. 2018, pp. 265–273. DOI: 10.1007/978-3-030-00934-2_30.

[122] Alexandra B Schroeder et al. "The ImageJ ecosystem: Open-source software for image visualization, processing, and analysis". In: *Protein Science* 30.1 (2021), 234–249.

[123] Will Schroeder, Kenneth M Martin, and William E Lorensen. *The visualization toolkit an object-oriented approach to 3D graphics*. Prentice-Hall, Inc., 1998.

[124] Bhumika Shokeen et al. "Omics and interspecies interaction". In: *Periodontology 2000* 85.1 (2021), pp. 101–111.

[125] Michael T Madigan; John M Martinko; Kelly S Bender; Daniel H Buckley; David A Stahl. *Brock biology of microorganisms*. 14th ed. Pearson, 2014. ISBN: 978-0-321-89739-8.

[126] Carsen Stringer et al. "Cellpose: a generalist algorithm for cellular segmentation". In: *Nature Methods* 18.1 (2021), 100–106.

[127] Patrick Taillandier et al. "Building, composing and experimenting complex spatial models with the GAMA platform". In: *GeoInformatica* 23.2 (2019), 299–322.

[128] Adrian Taruttis and Vasilis Ntziachristos. "Advances in real-time multispectral optoacoustic imaging and its applications". In: *Nature photonics* 9.4 (2015), 219–227.

[129] Philippe Thevenaz, Urs E Ruttimann, and Michael Unser. "A pyramid approach to subpixel registration based on intensity". In: *IEEE transactions on image processing* 7.1 (1998), 27–41.

[130] Jean-Yves Tinevez et al. "TrackMate: An open and extensible platform for single-particle tracking". In: *Methods* 115 (2017), 80–90.

[131] Seth Tisue and Uri Wilensky. "Netlogo: A simple environment for modeling complexity". In: *International conference on complex systems*. Vol. 21. Boston, MA. 2004, 16–21.

[132] Wen-Hsiang Tsai. "Moment-preserving thresolding: A new approach". In: *Computer vision, graphics, and image processing* 29.3 (1985), 377–393.

[133] Stratis Tzoumas et al. "Unmixing molecular agents from absorbing tissue in multispectral optoacoustic tomography". In: *IEEE transactions on medical imaging* 33.1 (2013), 48–60.

[134] Ryan S Udan et al. "Quantitative imaging of cell dynamics in mouse embryos using light-sheet microscopy". In: *Development* 141.22 (2014), 4406–4414.

[135] J Van Krugten, K-KH TARIS, and Erwin JG Peterman. "Imaging adult C. elegans live using light-sheet microscopy". In: *Journal of microscopy* 281.3 (2021), 214–223.

[136] Jaap Jan Vos et al. "Green light for liver function monitoring using indocyanine green? An overview of current clinical applications". In: *Anaesthesia* 69.12 (2014), 1364–1376.

[137] Stéfan van der Walt et al. "scikit-image: image processing in Python". In: *PeerJ* 2 (June 2014), e453. ISSN: 2167-8359. DOI: 10.7717/peerj.453. URL: https://doi.org/10.7717/peerj.453.

[138] Martin Weigert et al. "Content-aware image restoration: pushing the limits of fluorescence microscopy". In: *Nature methods* 15.12 (2018), pp. 1090–1097.

[139] Martin Weigert et al. "Star-convex Polyhedra for 3D Object Detection and Segmentation in Microscopy". In: *The IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2020. DOI: 10.1109/WACV45572.2020.9093435.

[140] Mark D Wilkinson et al. "The FAIR Guiding Principles for scientific data management and stewardship". In: *Scientific data* 3.1 (2016), 1–9.

[141] Ewa A Woznica et al. "Liver dysfunction in sepsis." In: *Advances in Clinical and Experimental Medicine: Official Organ Wroclaw Medical University* 27.4 (2018), 547–551.

[142] Mingxia Xie and Xinqiang Niu. "A 3D roaming and collision detection algorithm applicable for massive spatial data". In: *PLoS one* 15.2 (2020), e0229038.

[143] Jui-Cheng Yen, Fu-Juay Chang, and Shyang Chang. "A new criterion for automatic multilevel thresholding". In: *IEEE Transactions on Image Processing* 4.3 (1995), 370–378.

[144] Gregory W Zack, William E Rogers, and Samuel A Latt. "Automatic measurement of sister chromatid exchange frequency." In: *Journal of Histochemistry & Cytochemistry* 25.7 (1977), 741–753.

[145] Yingjie Zhu et al. "Evaluation of organ function in patients with severe COVID-19 infections". In: *Medicina Clinica* 155.5 (2020), 191–196.

Danksagung

Mit der Fertigstellung dieser Doktorarbeit möchte ich bei allen bedanken, die auf dem Weg hier hin unterstützend an meiner Seite gestanden sind.

Zuerst will ich mich bei meinem Doktorvater Prof. Dr. Marc Thilo Figge bedanken, der in mir das Interesse an dem Forschungsfeld der bildbasierten Systembiologie geweckt hat und mir später ermöglichte innerhalb der Arbeitsgruppe *Angewandte Systembiologie* zu promovieren. Seine exzellente Betreuung ermöglichte es mir meine Fähigkeiten weiterzuentwickeln und war geprägt von Offenheit und dem unkomplizierten Umgang bei etwaigen Schwierigkeiten oder Fragen. Dies erlaubte mir eigene Ideen für Lösungen und Konzepte zu entwickeln, für die er immer ein offenes Ohr hatte und, wenn nötig, auch kritisch hinterfragte. Insbesondere bin ich dankbar für die Erfahrungen außerhalb des regulären Wissenschaftsbetriebs, sei es bei der Organisation von Konferenzen, der Konzeption von Vorlesungen und Übungen, oder der Einrichtung von Webseiten und Webanwendungen.

Ich möchte mich auch bei den aktuellen und ehemaligen Mitarbeitenden der Arbeitsgruppe *Angewandte Systembiologie* bedanken, die immer ein angenehmes und offenes Arbeitsklima aufrechterhalten haben, wo jeder einem immer mit Rat, Tat und Literatur weiterhelfen konnte. Zu Beginn meiner Promotion haben mir insbesondere Anna Medyukhina und Alexander Tille geholfen mich zurechtzufinden und konnten mir hilfreiche Tipps zum Schreiben von Publikationen gegeben. Ich möchte mich auch bei Sandra Timme bedanken, die mir mit Informationen bezüglich des Aufbaus einer Dissertation sehr weitergeholfen hat. Insbesondere möchte ich meinem Büronachbarn Zoltán Cseresnyés für die lehrreiche Zusammenarbeit danken, bei der ich vieles über Mikroskopie, Bildanalyse, Halten von Vorlesungen, und dem Schreiben von wissenschaftlichen Texten lernen konnte. Hierbei möchte ich auch Mohamed I. Abdelwahab Hassan danken, welcher immer Antworten für biologischen Fragen und passende Literatur dazu hatte.

Auch meiner ehemaligen Kommilitonin Marie Lataretu möchte ich meinen Dank ausdrücken, da ich erst durch ihre Hilfe die Promotionsstelle gefunden habe. Auch bei meinen ehemaligen Kommilitonen und jetzt Mitarbeitern Bastian Seelbinder und Daniel Loos möchte ich für die interessanten Diskussionen und die Kinoabende danken.

Bedanken will ich mich auch bei meinen Freunden Klaus, Angelika, Sebastian, Danny, und allen anderen aus unserer kleinen Gruppe für die schönen Spieleabende und Unterstützung bei kleinen und größeren Problemen. Und im Speziellen will ich einer Person die nicht genannt werden will danken, die gerade in stressigen Zeiten im Studium und der Promotion immer ein offenes Ohr hatte.

Nicht zuletzt möchte ich mich auch herzlichst bei meinen Eltern bedanken, die immer an mich geglaubt und mir trotz Widerstand von anderen Leuten das Besuchen einer guten Schule und das Studium ermöglicht haben. Dieser ganze Weg bis hier hin war nur dank euch und eurer Liebe, eurem uneingeschränkten Rückhalt und eurer Unterstützung möglich. Und ich bin für all dies und mehr zutiefst dankbar!

Ehrenwörtliche Erklärung

Hiermit erkläre ich, dass mir die geltende Promotionsordnung der Fakultät für Biowissenschaften bekannt ist und ich mich mit bestem Wissen an diese Ordnung gehalten habe.

Die vorliegende Dissertation habe ich selbständig und nur unter Verwendung der angegebenen Hilfsmittel, Daten und Quellen angefertigt. Unterstützung während meiner wissenschaftlichen Arbeit und zur Erstellung des vorliegenden Dissertationstextes habe ich nur von den genannten Co-Autoren und in der Danksagung genannten Personen erhalten. Ich habe keine Hilfe von externen Vermittlungs- oder Beratungsdiensten in Anspruch genommen. Niemand hat mittelbare oder unmittelbare geldwerte Leistungen erhalten, für Arbeiten die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Die vorgelegte Dissertation wurde bisher nicht als Prüfungsarbeit für eine andere wissenschaftliche Prüfung eingereicht. Im Speziellen habe ich sie an keiner anderen Hochschule eingereicht, um einen akademischen Grad zu erhalten.

Ruman Gerst

Tabellarischer Lebenslauf

Ruman Gerst

Curriculum Vitae

Education

since 2018 **PhD Student**, *Applied Systems Biology research group, Leibniz Institute for Natural Product Research and Infection Biology – Hans-Knöll-Institute*, Jena/Germany
Title: Accessible software frameworks for reproducible image analysis of host-pathogen interactions
Supervisor: Prof. Dr. Marc Thilo Figge

2015 – 2017 **Master of Science (M.Sc.) Bioinformatics**, *Friedrich-Schiller-Universität Jena*, Jena/Germany

2012 – 2015 **Bachelor of Science (B.Sc.) Bioinformatics**, *Friedrich-Schiller-Universität Jena*, Jena/Germany

2003 – 2012 **Abitur**, *Melanchthon-Schule Steinatal*, Willingshausen/Germany

Theses

2017 **Master thesis**, *Bioinformatics/High-Throughput Analysis, Faculty of Mathematics and Computer Science, Friedrich Schiller University Jena*
Title: PCAGO: An interactive web service to analyze *RNA-Seq* data with principal component analysis
Supervisor: Prof. Dr. Manja Marz

2015 **Bachelor thesis**, *Bioinformatics/High-Throughput Analysis, Faculty of Mathematics and Computer Science, Friedrich Schiller University Jena*
Title: Statistical analysis of relation between RNA coding sequence and secondary structure
Supervisor: Prof. Dr. Manja Marz

Professional experience

11/2021 **Co-organizer**, *6th International Symposium on Systems Biology of Microbial Infections*, Jena/Germany

10/2020 **Co-organizer**, *5th International Symposium on Image-based Systems Biology*, Jena/Germany

since 2018 **Doctoral researcher**, *Applied Systems Biology research group, Leibniz Institute for Natural Product Research and Infection Biology – Hans-Knöll-Institute*, Jena/Germany

Publications

Papers

[1] Gerst R, Cseresnyés Z, & Figge MT: **JIPipe: Visual batch processing for ImageJ.** Preprint: *ResearchSquare* `https://dx.doi.org/10.21203/rs.3.rs-1641739/v1` [Submitted to Nature Methods, May 2022]

[2] Hoffmann B, Gerst R, Cseresnyés Z, Foo W, Sommerfeld O, Press AT, Bauer M, & Figge MT: **Spatial quantification of clinical biomarker pharmacokinetics through deep learning-based segmentation and signal-oriented analysis of MSOT data.** *Photoacoustics* 2022, 100361.

[3] Büttner H, Niehs SP, Vandelannoote K, Cseresnyés Z, Dose B, Richter I, Gerst R, Figge MT, Stinear TP, Pidot SJ, & Hertweck C: **Bacterial endosymbionts protect beneficial soil fungus from nematode attack.** *Proceedings of the National Academy of Sciences* 2021, 118(37).

[4] Gerst R, Medyukhina A, & Figge, MT: **MISA++: A standardized interface for automated bioimage analysis.** *SoftwareX* 2020, 11, 100405.

Talks

[1] Gerst R, Cseresnyés Z, Figge MT: **JIPipe: Building automated image analysis tools without programming.** *International Joint Meeting Infection Biology and Antibiotics* (2022), *Berlin/Germany*

[2] Gerst R, Cseresnyés Z, Figge MT: **Designing fully automated bioimage processing pipelines without programming.** *International Conference on Medical Imaging Science and Technology* (2021), *Shenzen/China (Online)*

[3] Gerst R, Praetorius JP, Cseresnyés Z, Figge MT: **JIPipe: Visual programming for ImageJ.** *BioImage Informatics 2021* (2021), *Paris/France (Online)*

[4] Gerst R, Praetorius JP, Cseresnyés Z, Figge MT: **Bioimage analysis with deep learning for everyone: Visual programming in JIPipe.** *German Conference on Bioinformatics 2021* (2021), *Frankfurt am Main/Germany (Online)*

[5] Gerst R, Praetorius JP, Cseresnyés Z, Figge MT: **JIPipe: Designing image analysis pipelines without programming.** *8th International Conference on Microbial Communication for Young Scientists* (2021), *Jena/Germany (Online)*

[6] Gerst R, Cseresnyés Z, Figge MT: **JIPipe: a graphical batch-processing language for ImageJ.** *5th International Symposium on Image-based Systems Biology* (2020), *Jena/Germany (Online)*

[7] Gerst R, Medyukhina A, Figge MT: **MISA++: A high-performance framework for automated analysis of big volume image data.** *ILRS Joint Meeting 2019* (2019), *Wittenberg/Germany*

[8] Gerst R, Medyukhina A, Figge MT: **MISA++: A modular and high-performance framework for the analysis of light sheet microscopy data.** *Swiss Light-sheet Microscopy Workshop 2019* (2019), *Zurich/Switzerland*

[9] Gerst R, Medyukhina A, Figge MT: **Towards an open high-performance platform for fully-automated analysis of whole organ lightsheet fluorescence microscopy data.** *4th International Symposium on Image-based Systems Biology* (2018), *Jena/Germany*

Posters

[1] Gerst R, Cseresnyés Z, Figge MT: **JIPipe: Designing automated image analysis pipelines without programming.** *elmi2021* (2021), *Oxford/United Kingdom (Online)*

[2] Gerst R, Cseresnyés Z, Figge MT: **JIPipe: a graphical batch-processing language for ImageJ.** *5th International Symposium on Image-based Systems Biology* (2020), *Jena/Germany (Online)*

[3] Gerst R, Medyukhina A, Figge MT: **MISA++: a modular and high-performance framework for analysis of light sheet microscopy images.** *5th International Symposium on Image-based Systems Biology* (2020), *Jena/Germany (Online)*

[4] Gerst R, Medyukhina A, Figge MT: **MISA++: a standardized interface for automated high- performance big volume image analysis.** *NEUBIAS 2020* (2020), *Bordeaux/France*

[5] Gerst R, Medyukhina A, Figge MT: **Towards an open high-performance platform for fully-automated analysis of whole organ lightsheet fluorescence microscopy data.** *4th International Symposium on Image-based Systems Biology* (2018), *Jena/Germany*

[6] Gerst R, Medyukhina A, Figge MT: **Fast and efficient fully automated processing of lightsheet microscopy data to evaluate the glomerular number in kidneys.** *11th ILRS Symposium* (2018), *Jena/Germany*

Ruman Gerst

Anhang

FORMULAR 2

Manuskript Nr. 1

Kurzreferenz Hoffmann et al. (2022), Photoacoustics

Beitrag des Doktoranden / der Doktorandin

Beitrag des Doktoranden / der Doktorandin zu Abbildungen, die experimentelle Daten wiedergeben (nur für Originalartikel):

Figures 1-7; Supplementary Figures A1, A4-A8, A10	☐	100 % (die in dieser Abbildung wiedergegebenen Daten entstammen vollständig experimentellen Arbeiten, die der Kandidat/die Kandidatin durchgeführt hat)
	☒	0 % (die in dieser Abbildung wiedergegebenen Daten basieren ausschließlich auf Arbeiten anderer Koautoren)
	☐	Etwaiger Beitrag des Doktoranden / der Doktorandin zur Abbildung: _____% Kurzbeschreibung des Beitrages: *(z. B. „Abbildungsteile a, d und f“ oder „Auswertung der Daten“ etc)*
Figure 8; Supplementary Table A2	☒	100 % (die in dieser Abbildung wiedergegebenen Daten entstammen vollständig experimentellen Arbeiten, die der Kandidat/die Kandidatin durchgeführt hat)
	☐	0 % (die in dieser Abbildung wiedergegebenen Daten basieren ausschließlich auf Arbeiten anderer Koautoren)
	☐	Etwaiger Beitrag des Doktoranden / der Doktorandin zur Abbildung: _____% Kurzbeschreibung des Beitrages: *(z. B. „Abbildungsteile a, d und f“ oder „Auswertung der Daten“ etc)*

_____ _____

Unterschrift Kandidat/-in Unterschrift Betreuer/-in (Mitglied der Fakultät)

FORMULAR 2

Manuskript Nr. 2

Kurzreferenz Gerst et al. (2022), Research Square (*Preprint*)

Beitrag des Doktoranden / der Doktorandin

Beitrag des Doktoranden / der Doktorandin zu Abbildungen, die experimentelle Daten wiedergeben (nur für Originalartikel):

Figure 1-2; Supplementary Figures 1.1, 3.1-3.7, Supplementary Table 5.1	☒	100 % (die in dieser Abbildung wiedergegebenen Daten entstammen vollständig experimentellen Arbeiten, die der Kandidat/die Kandidatin durchgeführt hat)
	☐	0 % (die in dieser Abbildung wiedergegebenen Daten basieren ausschließlich auf Arbeiten anderer Koautoren)
	☐	Etwaiger Beitrag des Doktoranden / der Doktorandin zur Abbildung: _____% Kurzbeschreibung des Beitrages: *(z. B. „Abbildungsteile a, d und f" oder „Auswertung der Daten" etc)*
Figure 3	☐	100 % (die in dieser Abbildung wiedergegebenen Daten entstammen vollständig experimentellen Arbeiten, die der Kandidat/die Kandidatin durchgeführt hat)
	☐	0 % (die in dieser Abbildung wiedergegebenen Daten basieren ausschließlich auf Arbeiten anderer Koautoren)
	☒	Etwaiger Beitrag des Doktoranden / der Doktorandin zur Abbildung: 50 % Kurzbeschreibung des Beitrages: *Layout der Abbildung*
Supplementary Figures 2.1-2.5	☐	100 % (die in dieser Abbildung wiedergegebenen Daten entstammen vollständig experimentellen Arbeiten, die der Kandidat/die Kandidatin durchgeführt hat)
	☒	0 % (die in dieser Abbildung wiedergegebenen Daten basieren ausschließlich auf Arbeiten anderer Koautoren)
	☐	Etwaiger Beitrag des Doktoranden / der Doktorandin zur Abbildung: _____% Kurzbeschreibung des Beitrages:

_____ _____

Unterschrift Kandidat/-in Unterschrift Betreuer/-in (Mitglied der Fakultät)

FORMULAR 2

Manuskript Nr. 3

Kurzreferenz Büttner et al. (2021), PNAS

Beitrag des Doktoranden / der Doktorandin

Beitrag des Doktoranden / der Doktorandin zu Abbildungen, die experimentelle Daten wiedergeben (nur für Originalartikel):

Figures 1-3; Tables S1-S9; Figures S1-S22	☐	100 % (die in dieser Abbildung wiedergegebenen Daten entstammen vollständig experimentellen Arbeiten, die der Kandidat/die Kandidatin durchgeführt hat)
	☒	0 % (die in dieser Abbildung wiedergegebenen Daten basieren ausschließlich auf Arbeiten anderer Koautoren)
	☐	Etwaiger Beitrag des Doktoranden / der Doktorandin zur Abbildung: _____% Kurzbeschreibung des Beitrages: *(z. B. „Abbildungsteile a, d und f“ oder „Auswertung der Daten“ etc)*

_____ _____

Unterschrift Kandidat/-in Unterschrift Betreuer/-in (Mitglied der Fakultät)

FORMULAR 2

Manuskript Nr. 4

Kurzreferenz Gerst et al. (2020), SoftwareX

Beitrag des Doktoranden / der Doktorandin

Beitrag des Doktoranden / der Doktorandin zu Abbildungen, die experimentelle Daten wiedergeben (nur für Originalartikel):

Figures 1-2, Figure 4; Table S1-S4; Figures S3-S5; Figures S8-S10; Listings S1-S11	☒	100 % (die in dieser Abbildung wiedergegebenen Daten entstammen vollständig experimentellen Arbeiten, die der Kandidat/die Kandidatin durchgeführt hat)
	☐	0 % (die in dieser Abbildung wiedergegebenen Daten basieren ausschließlich auf Arbeiten anderer Koautoren)
	☐	Etwaiger Beitrag des Doktoranden / der Doktorandin zur Abbildung: _____% Kurzbeschreibung des Beitrages:
Figure 3; Figures S6-S7	☐	100 % (die in dieser Abbildung wiedergegebenen Daten entstammen vollständig experimentellen Arbeiten, die der Kandidat/die Kandidatin durchgeführt hat)
	☐	0 % (die in dieser Abbildung wiedergegebenen Daten basieren ausschließlich auf Arbeiten anderer Koautoren)
	☒	Etwaiger Beitrag des Doktoranden / der Doktorandin zur Abbildung: 75 % Kurzbeschreibung des Beitrages: *Layout der Abbildung*

_____ _____

Unterschrift Kandidat/-in Unterschrift Betreuer/-in (Mitglied der Fakultät)