GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Master's Thesis

submitted in partial fulfillment of the
requirements for the course "Applied Computer Science"

Optimisation of microscopy image analysis with HPC

Sören Wieduwilt

Institute of Computer Science

Bachelor's and Master's Theses
of the Center for Computational Sciences
at the Georg-August-Universität Göttingen

02. January 2026

Georg-August-Universität Göttingen
Institute of Computer Science

Goldschmidtstraße 7
37077 Göttingen
Germany

☎ +49 (551) 39-172000
FAX +49 (551) 39-14403
✉ office@informatik.uni-goettingen.de
🌍 www.informatik.uni-goettingen.de

First Supervisor:	Prof. Dr. Julian Kunkel
Second Supervisor:	Prof. Dr. Constantin Pape

Declaration on the use of ChatGPT and comparable tools in the context of examinations

In this work I have used ChatGPT or another AI as follows:

- ☐ Not at all
- ☐ During brainstorming
- ☐ When creating the outline
- ☐ To write individual passages, altogether to the extent of 0% of the entire text
- ☐ For the development of software source texts
- ☐ For optimizing or restructuring software source texts
- ☑ For proofreading or optimizing
- ☐ Further, namely: -

I hereby declare that I have stated all uses completely.
Missing or incorrect information will be considered as an attempt to cheat.

Abstract

Modern high-resolution microscopes produce large amounts of image and metadata. To manage the data and make it accessible for researchers and medical staff, Research Data Management (RDM) systems are utilised. These are capable of managing users and groups, enabling collaboration between them, managing access controls, and handling large data repositories. To extract insights and knowledge from the large amount of data, computer-aided analysis workflows are needed.

This thesis designs a connection pipeline between the microscopy RDM system Open Microscopy Environment Remote Objects (OMERO) and the computing resources of a modern High-Performance Computing (HPC) system. This allows researchers and medical staff to directly start analysis steps from the browser without having to learn the usage of HPC job executions. This enables more users to integrate HPC usage into their scientific workflows.

For this, an existing analysis workflow, CellDetector, is evaluated. The requirements are grouped into five categories: performance, usability, security, scalability, and transferability. Based on the requirements, a pipeline design is created that establishes a secure connection between the OMERO system environment and the HPC system environment. The connection utilises the SSH forced command mechanism to limit access to the HPC system, securing both the user space and the HPC system. Additional integrated user interfaces are created to initiate the analysis steps and improve usability.

After implementing the pipeline with the CellDetector workflow, the system is evaluated. The updated workflow shows improvements in performance and usability while remaining secure and scalable. Furthermore, the proposed pipeline design is transferable to similar system environments.

Acknowledgments

I would like to thank my supervisors, Prof. Dr. Julian Kunkel and Prof. Dr. Constantin Pape, for introducing me to the topic of this thesis and giving me valuable guidance and feedback.

Furthermore, I would like to express my gratitude to Dr. Hendrik Nolte and Dr. med. Jonas Franz for their continuous support by exchanging ideas, giving me feedback on the software and the thesis, and allowing me to use the CellDetector workflow.

Finally, I would like to thank my family for their support and encouragement during all this time.

Contents

List of Figures

List of Tables

List of Listings

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
BIOMERO	BioImage analysis in OMERO
CA	Certificate Authority
CFCI	Core Facility Cellular Imaging
CLI	Command Line Interface
CPU	Central Processing Unit
CSV	Comma Seperated Values
CT	Computed Tomography
CUDA	Compute Unified Device Architecture
DAG	Directed Acyclic Graph
DSRP	Design Science Research Process
eCDF	empirical Cumulative Density Function
FAIR	Findability, Accessibility, Interoperability, and Resusability
GPGPU	General-Purpose Computing on Graphics Processing Units
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HIP	Heterogeneous-Compute Interface for Portability
HPC	High-Performance Computing
HTML	Hypertext Markup Language
IDR	Image Data Resource
JSON	JavaScript Object Notation
GB	Gigabyte
GWDG	Gesellschaft für wissenschaftliche Datenverarbeitung mbH
HDD	Hard Disk Drive
HEAppE	High-End Application Execution Middleware

ISO	International Organization for Standardization
KPI	Key Performance Indicator
LDAP	Lightweight Directory Access Protocol
MB	Megabyte
MPI	Message Passing Interface
MVC	Model-View-Controller
NCEM	National Center for Electron Microscopy
NERSC	National Energy Research Scientific Computing
MRI	Magnetic Resonance Imaging
NN	Neural Network
OAuth	Open Authorization
OMERO	Open Microscopy Environment Remote Objects
OpenACC	Open Accelerators
OS	Operating System
RDBMS	Relational Database Management Software
RDM	Research Data Management
REST	Representational State Transfer
ROI	Regions of Interest
SCP	Secure Copy
SIMD	Single Instruction, Multiple Data
SSH	Secure Shell
SSL	Secure Sockets Layer
TB	Terabyte
TCP	Transmission Control Protocol
UMG	Universitätsmedizin Göttingen
URL	Uniform Resource Locator
VM	Virtual Machine
WSI	Whole Slide Imaging
ZeroMQ	Zero Message Queue

Chapter 1

Introduction

Over the past few years, many research areas have experienced a significant increase in the volume of data generated. As a result, the demand for analysis and processing capabilities for large datasets is also increasing [1]. High-Performance Computing (HPC) systems can help researchers process these large amounts of data within reasonable timeframes [2]. However, connecting existing systems to HPC resources can be cumbersome and complicated for users, disrupting established workflows [3].
One research area that produces large amounts of data in the form of high-resolution microscopy images, in various formats and sizes, is neuropathology [4]. To extract knowledge from these images, the often large datasets must be analyzed promptly or prepared for further processing.
The goal of this thesis is to develop a pipeline design proposal to connect the open-source Research Data Management (RDM) system, Open Microscopy Environment Remote Objects (OMERO), with HPC resources and to create support for image analysis. In this use case, the OMERO system used at the Universitätsmedizin Göttingen (UMG) will be connected to the HPC system at the Gesellschaft für wissenschaftliche Datenverarbeitung mbH (GWDG) to improve existing workflows.
OMERO supports a wide range of microscopy file formats from various manufacturers, making it a suitable system for the centralized storage and management of images [5]. To support the staff at the UMG, a workflow was created to analyse tissue sample images and detect cells. In the next step, the analysis results are used to further train a pre-trained Artificial Intelligence (AI) model, thereby creating an image recognition model for specific cell types. The AI model can then be used on new images to assist the user in identifying and quantifying these cell types in parts of the tissue sample and to recognise the state of a medical condition of a patient or use the analysis for research.

1.1 Goal Definition

The current pipeline has certain limitations that require improvement. The primary challenges are the long processing times and the system's complexity for users without a background in computer science.
From these challenges, several Key Performance Indicators (KPIs) can be derived:

- Performance: Wall-clock time required for the different processing steps to complete successfully.
- Usability: User experience when working through the workflow steps.
- Security: System and data security when connecting different system environments.
- Scalability: Accommodate increasing numbers of users and data.
- Transferability: Extending the pipeline design to different workflows and being able to apply the pipeline approach to other similar system environments.

These challenges give rise to several research questions, which will be addressed in this thesis and are presented in Table 1.1.

Name	Research Question
Q1	What are the options for improving the performance of analysing and processing microscope images by utilising HPC resources?
Q2	How can the operation of the HPC resources in this workflow be made more user-friendly?
Q3	What potential security issues could arise, and how could they be addressed or minimised?
Q4	What aspects need to be considered as the amount of data, number of users, and future workflows grow?

Table 1.1: Research Questions

1.2 Research Methodology

The research methodology used for this thesis is the Design Science Research Process (DSRP) Model [6] which is a Design Science approach for information systems. Similar to other Design Science Research methodologies, DSRP involves the design of an artifact intended to address a particular problem, followed by an evaluation of the artifact. The six activities from the model will be followed and evaluated based on the described use case. The activities are as follows:

1. **Problem identification and motivation**: This activity analyzes the problem of the use case and how the solution can offer value.

2. **Objectives of a solution**: The objectives of the work will be identified by examining the stated problems and reviewing related research.
3. **Design and development**: This activity describes the design and implementation of the desired artifact. It includes the functionality and architecture, as well as the process by which the design solution is developed.
4. **Demonstration**: The completed artifact will be tested in the context of the use case.
5. **Evaluation**: The results will be assessed and evaluated.
6. **Communication**: The thesis, along with the discussions with potential users about the solution, represent the communication.

1.3 Contributions

By addressing the research questions, several contributions will be presented in the following thesis:

- Proposing an improved pipeline design based on the requirements and an analysis of the current system.
- Improvement of performance to allow for a more efficient workflow.
- The workflow's usability will be improved to make it more accessible to a larger group of users.
- Integration of a secure communication method and secure data exchange between the systems.
- The scalability and transferability of the systems in the pipeline, with regard to future workflows, will be ensured.

1.4 Structure

The thesis consists of seven chapters and is structured as follows:

- **Chapter 2**: The current state of the existing system is analysed and a requirements analysis is conducted. Based on the requirements, the main KPIs are identified and prioritised.
- **Chapter 3**: In this chapter, background information about the data management system OMERO and the topic of HPC is provided. The related literature is analysed.
- **Chapter 4**: The general pipeline design is developed by evaluating options and proposing a general system architecture.

- **Chapter 5**: Based on the design concepts and requirements, an implementation of the pipeline in the context of the use case is conducted.
- **Chapter 6**: The workflow is evaluated with the help of the identified KPIs.
- **Chapter 7**: This chapter draws a conclusion and proposes topics for further research.

Chapter 2

Current State and Requirement Analysis

To improve and expand the current workflow of the use case, the requirements of such a system have to be identified. To do this, Section 2.1 gives a few insights into the use case. Section 2.2 describes the current state of the running workflow and in Section 2.3 a requirement analysis is done. With these results, multiple KPI can be derived to measure the changes.

This thesis is about connecting the RDM system OMERO with HPC resources to improve microscopy image analysis. To do this, an existing workflow will be adjusted and integrated. The system and workflow this thesis describes and wants to improve, is created by Dr. med. Jonas Franz from the UMG and the Institute of Neuropathology. The existing system is presently in use at the UMG and connects to ressources of the GWDG. In the context of this thesis a limited set of research data is used for improving the system.

2.1 Use Case Description

The Institute of Neuropathology employs both traditional and digital microscopy for diagnostic and research purposes. One technology, which is part of digital microscopy, is Whole Slide Imaging (WSI). To enhance the diagnosis quality and accelerate research, Whole Slide Images are widely used in the field of neuropathology [4]. WSI is often called "virtual microscopy" [7] and it tries to improve on conventional light microscopy through a computer-generated model [8]. WSI has several advantages over traditional microscopy but also several limitations. The advantages include workflow improvements, like remote access to the images, sharing of images and simultaneous access. Furthermore the image and staining quality does not deteriorate over time [8]. The digitisation of the images also enables computer-aided analysis, like in this use case. The limitations and challenges include, the high initial cost for the scanner, storage and network infrastructure and possible image fidelity differences [9]. To obtain a WSI, several steps have to be performed in the laboratory of the UMG.

Figure 2.1 illustrates a typical workflow required to create WSIs. The steps can vary depending on factors like chemicals used, time, temperature and many others. At first, a tissue sample must be collected. In neuropathological diagnostics the sample is obtained through a biopsy or in the case of a deceased person, during an autopsy examination. To fixate the tissue sample and preserve the cellular structure, a fixative, such as phosphate formalin, is used. The fixative depends on the tissue type and the time in this process can influence the staining step. Afterwards, the fixative and other fluids are cleared with alcohol, which is subsequently removed. In the next step, the specimen is embedded in a supporting material, such as paraffin wax, chilled, and then sectioned into thin slices using a microtome. Any folds are afterwars removed with the aid of a water bath. To highlight certain components of the specimen, chemical compounds are added as dyes. The chemical reactions of these compounds provide valuable information for an analysis. The prepared samples are covered with a mounting media and protected by a glass cover. The final step involves scanning the specimen using a WSI scanner. The scanner is capable of operating in bright-field mode, using fluorescence illumination, or combining both techniques. Different scales and magnification levels can be applied during scanning. Various WSI scanner vendors employ distinct file formats and metadata conventions [10].

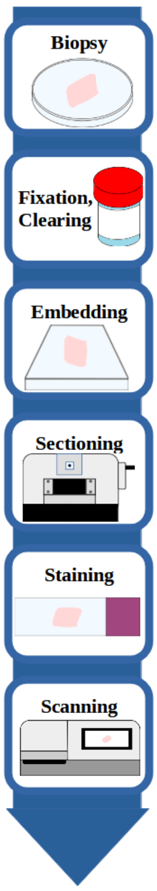


Figure 2.1: WSI acquisition steps

Figure 2.2 shows an example of a WSI in different magnification levels and with two highlighted dye channels. The green channel, fluorescein isothiocyanate (FITC), is used to visualise certain proteins and cellular structures, while the blue channel, 4,6-diamidino-2-phenylindole (DAPI), is used to visualise cell nuclei as it binds to DNA. The digitised images can subsequently be used for visual observations as well as for computer-assisted analysis. High-resolution images are typically multiple gigabytes in storage size and some can reach up to 40 gigabyte [11] [12]. To visualise and manage the images, vendor tools or open-source tools, such as OMERO, are used. To store the images, sufficient storage capacity in a data repository must be reserved.

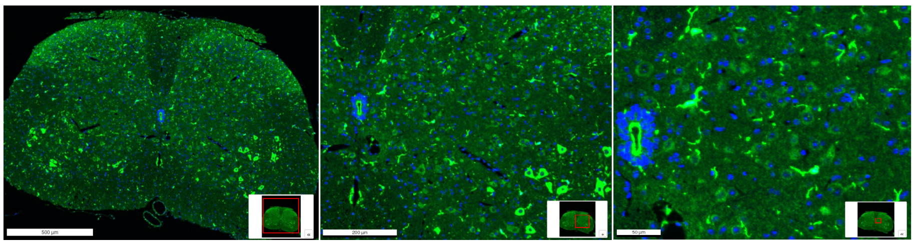


Figure 2.2: Example: WSI of a mouse tissue section

The main goal of this project is to design a pipeline for microscopy analysis workflows based on the existing approach. The design choices aim to enable broader adoption of these workflows by researchers and medical staff.

The workflow is part of the project "CellDetector: Deep transfer learning based classification of single cells in tissue" by Dr. med. Jonas Franz [13]. Regions of Interest (ROIs) of the tissue are annotated on the WSI in OMERO. With these WSI a nuclei detection algorithm and image analysis is started. Based on the results of this step, the pre-sorted cells must be annotated in a web application. The images and corresponding annotations are used to train a neural network (NN). The resulting model can then be used to support users in subsequent research and analysis steps.

2.2 Current State

The **architecture** of the current system consists of several components of the UMG and GWDG and is illustrated in Figure 2.3. The source of the WSI are the WSI-Scanners of the UMG and their image repositories. The images originate from different teams and include important research data and/or highly sensitive health data of patients, and therefore have to be handled confidentially and securely. Workstations are used to analyse the images and to prepare the image data for further processing.
The current workflow depicted in this thesis uses the open-source RDM system OMERO. An OMERO server instance is deployed across two virtual servers provided by GWDG. One virtual server hosts the OMERO.web software, while the other hosts the OMERO.server software and the data repository. Two additional virtual servers from the GWDG are used for the image processing steps. One server functions as a relational database server running PostgreSQL, while the second server operates as a web server using Jupyter Notebook with the Voilà extension to create a web application. To gain information from the WSI, the images are analysed using the HPC system of the GWDG. The connection to the HPC login node is via Secure Shell (SSH) and the associated public-private key encryption over a shell user interface.

The current **workflow** consists of **three steps**:

- **Preprocessing**: Analysing the images and preparing the necessary data for the training step.
- **Neural Network Training**: Training a NN model with the data of the preprocessing step.
- **Neural Network Prediction**: Using the NN model on new and existing image data to help the medical/research staff.

Step 1: Preprocessing: The preprocessing step is needed to analyse the images and gather the data needed to train the NN model. The preprocessing itself consists of three steps. For this part of the workflow, the user utilises Apache Airflow. Apache Airflow is an open-source workflow management software and is used for data engineering pipelines [14]. The data pipelines are depicted as Directed Acyclic Graphs (DAGs) and consist of nodes and connections. The nodes

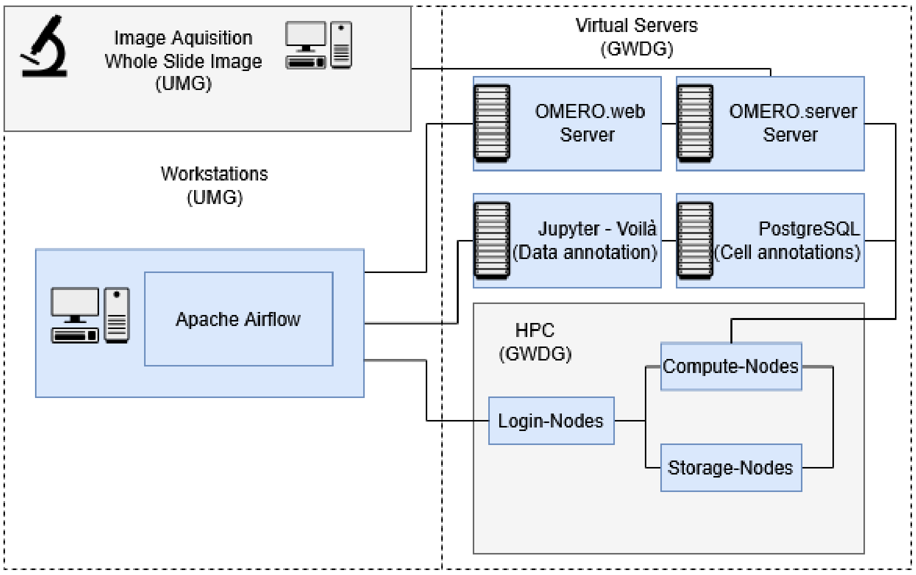


Figure 2.3: Overview of the system architecture

represent tasks which can be code (e.g. Python, Bash) and the connections represent the order in which the tasks are executed. The DAGs themselves are written in Python. One DAG represents the preprocessing steps and executes these automatically in the right order after the DAG is started. First, the user gathers the image IDs in OMERO using the web interface (OMERO.web) and prepares a list of these IDs in the form of a Comma-Separated Values (CSV) file. This list serves as the input for the preprocessing DAG. Another important input for the preprocessing step are the ROIs. The user has to annotate specific areas in the WSI to specify and differentiate parts of the sample (e.g. tissue, hole). These annotations are performed in the OMERO.web interface. The output of the DAG consists of files required to initiate work on the HPC and to prepare for the subsequent steps of the workflow. The necessary files have to be transferred over the the default SSH port 22 to the HPC-system and saved on its storage nodes. On the HPC login nodes, processing of the WSI can be initiated using the Slurm Workload Manager. The Slurm Workload Manager is an open-source job-scheduler, that distributes workloads across the HPC compute nodes [15]. The three scripts of the preprocessing step are started, with a Slurm template, in a fixed order.

1. **Zarr file creation:** The first script uses the Python Application Programming Interface (API), OMERO.py, to connect to the OMERO.server. By default, this connection uses port 4064 on the OMERO server and operates over the Transmission Control Protocol (TCP). The script transfers the pixel data of the selected WSI and creates a new Zarr file with this data. The Zarr file format is an open standard for the storage of large multidimensional arrays. The Zarr standard supports high-throughput distributed I/O on different storage systems. A Zarr file is a directory-based file in the form of a grid of chunks, each as its own file,

that store the multidimensional array and additional metadata files, that describe the file. This format enables random access to subsets of huge datasets and allows for parallel and distributed workflows [16]. The chunks can be compressed to reduce storage requirements, and programs can read and write only the necessary chunks, thereby reducing memory usage. The pixel data of the WSI is then stored as a Zarr file on the HPC storage for further analysis steps.

2. **Cellpose:** In the next step, the Python script retrieves the pixel data of the WSI from the OMERO.server and prepares it for the initial image processing step. To extract information from the raw WSI, it is necessary to differentiate objects within the image data. This process is often performed manually by research members and medical staff, but it can be accelerated with the use of image segmentation tools. One such a tool is Cellpose, a generalist algorithm for cellular segmentation [17]. It is a deep learning-based model that is trained on over 70,000 segmented objects of cells. It supports 2D and 3D image data. Cellpose differentiates and segments cell bodies, membranes and nuclei. With the the prepared WSI and Cellpose the script analyses the image data for cell nuclei by using the deep learning model. The output consists of a list containing the results from Cellpose, along with additional image data statistics required for the subsequent steps. This list is stored on the HPC storage and is uploaded to the OMERO.server with the OMERO Command Line Interface (OMERO.cli) and is attached to the image as an annotation file.

3. **empirical Cumulative Density Function (eCDF):** The final step of preprocessing involves calculating the eCDF over the ROIs defined in OMERO. The script uses the OMERO.py API to connect to OMERO and get the polygonal regions. The Zarr file stored on the HPC is used for this step. For each distinct region, the eCDF is calculated. The eCDF is an empirical distribution function that measures what fraction of the data is less or equal to a given value and is in this case used to analyse background noise and colors of the regions of the image. The outputs of this step are the eCDF values, which are saved on the HPC storage and uploaded to the OMERO.server using OMERO.cli, where they are attached to the images as annotation files.

Step 2: Neural Network Training: The next step of the workflow is the annotation of the cells for the training of the NN. For this step the current system uses Jupyter Voilà as a web application. Jupyter Notebook is an open-source tool that integrates code, text, and visualizations in a single document and is frequently used in teaching and research [18]. Jupyter Voilà is an extension to Jupyter Notebook and turns a Jupyter Notebook into a web application. This web application is the user interface to annotate the cells. To reduce the number of cells that must be annotated, a preselection is performed using the additional results from the preprocessing steps. This process filters out cells that are less likely to belong to the target cell type, intentionally introducing a bias into the dataset. Each user loads its own generated data and sees multiple images of one cell candidate in different perspectives. The user can then decide whether a cell belongs to the

selected cell type, does not belong, or should be skipped. After submission, the next candidate cell is presented. The cell annotations are stored for each user on the PosgreSQL server.
With the results of the cell annotation and the second DAG, a pretrained ResNet-50 model is trained on the HPC compute nodes with the open-source deep learning framework PyTorch.

Step 3: Neural Network Prediction: The final step of the workflow is to apply the trained NN model on existing and new WSIs. For this purpose, a prediction Python script is executed on the HPC compute nodes, using the outputs of the Cellpose step to classify the cells. The predictions of the classification are stored on the HPC storage.
The third and fourth DAG are used to visualise the results in OMERO.web. It duplicates the WSI in OMERO and creates ROIs which indicate all predicted Cells and their percentages. Additionally, statistics are computed, such as the cell density within a specific ROI.

The system has currently one user that utilises the whole workflow, Dr. med. Jonas Franz, and several users that are only utilising a small part of the workflow, like the annotation of cell images. All users are part of the Institute of Neuropathology. Depending on the WSIs, the number of WSIs and on the storage size of the WSIs, the processing of the workflow can take multiple days of compute time. Based on these observations of the current system, a set of requirements can be derived to improve the workflow.

2.3 Requirement Analysis

To create a transferable pipeline, that connects the RDM system OMERO with HPC resources, it is necessary to define requirements for the design process and analyse the constraints. The requirement gathering is split into two phases, the requirement elicitation and the requirement analysis. The requirement elicitation is about gathering the raw requirements of the stakeholders and discussing their needs [19]. A part of this elicitation phase is already presented in Section 2.2 as part of the overview of the current system. The requirements could be gathered from observing the workflow, looking at the existing code repositories and in discussions with the project manager, Dr. med. Jonas Franz. These requirements consist of a number of functional and non-functional requirements.

Constraints: The requirements that define boundaries for the system and the design of the solution are called constraint requirements and are shown in Table 2.1.
The goal of the thesis sets key requirements, that translate to constraint requirements. For this use case it is a requirement to utilise the existing OMERO system and its data repository. The workflow is also already working and should be adapted to the new design proposal, instead of rewriting it. Developing a pipeline between OMERO and HPC resources also requires an HPC connection between those two.

Requirement ID	Description
CR-01	The RDM system OMERO shall be used as data repository and frontend Web application.
CR-02	The existing OMERO.server, OMERO.web and the existing data repository shall be used.
CR-03	The existing Python code base for the workflow should be used and adapted.
CR-04	The compute-intensive processes shall run on a HPC system.
CR-05	The OMERO.server and the HPC system shall be connected to run processing steps.

Table 2.1: Requirements: Constraints

Usability: One important goal of this project is to enable more users to execute more parts of the workflow, especially concerning the connection to the HPC resources. Table 2.2 describes the requirements connected to the usability of a new solution.

Requirement ID	Description
UR-01	One or multiple graphical user interfaces (GUI) shall be developed for parts of the workflow and replace the need for a shell user interface.
UR-02	The GUI shall connect to the data of the OMERO.server.
UR-03	The GUI should be started from the OMERO.web interface.
UR-04	The GUI shall connect OMERO to the HPC resources.

Table 2.2: Requirements: Usability

A challenge of the current system is, that most of the possible users do not have much experience with HPC systems and shell user interfaces. The requirements for creating a GUI aim to make the user experience more accessible and to remove potential barriers for new users. By integrating the GUI into OMERO.web, the process is further streamlined.

Performance: The steps described in Section 2.2 are computationally intensive and benefit from scalable HPC resources. This results in the requirements shown in Table 2.3

Requirement ID	Description
PR-01	Compute bottlenecks shall be identified.
PR-02	I/O bottlenecks shall be identified.
PR-03	Network bottlenecks shall be identified.
PR-04	The performance/efficiency of the largest bottlenecks should be improved.

Table 2.3: Requirements: Performance

HPC resources are suitable for scalable workloads. By designing a connection to the HPC system, it is important to understand the current limitations of the current code base, system architecture and data management and search for solutions to improve the utilisation of the HPC resources. If the analysis identifies any areas for improvement, they should be addressed.

Security: The management of sensitive image data and the secure connection to the HPC system have to be taken into account. Table 2.4 lists the requirements connected to security.

Requirement ID	Description
SeR-01	Management, transfer and processing of the WSI shall be secure and confidential.
SeR-02	The connection to the HPC shall comply with the security standards of the HPC.
SeR-03	Access to the OMERO server and its data repository shall be secure.

Table 2.4: Requirements: Security

The WSIs represent scientific images, data, and patient information. Therefore, it must be ensured that only the responsible scientific and medical staff have access to this data. The access has to be limited for the entire data lifecycle in this workflow. Furthermore, connecting two different systems, such as the virtual servers and the HPC system, can introduce additional attack vectors. It is important to identify these vulnerabilities and mitigate or minimize the associated risks.

Scalability: Table 2.5 lists the requirements that allow the system to reach a broader user base by scaling up its technical capacity.

Requirement ID	Description
ScR-01	The data repository should be able to scale according to the demand.
ScR-02	The Web application should be able to scale with an increase in users.
ScR-03	The processing infrastructure should be able to support an increase in workloads.

Table 2.5: Requirements: Scalability

The WSI require sufficient storage capacity and the demand increases with each new user and team. The system should be able to adapt accordingly. The same considerations apply to the user-oriented web interface and the processing infrastructure.

Transferability: The design of this use case should be applicable to other similar workflows. The transferability requirements can be seen in Table 2.6.
The pipeline developed in this thesis is designed to enable similar workflows, using OMERO as the data management system and an HPC system for computationally intensive processing steps.

Requirement ID	Description
TR-01	The new GUI design shall be transferable to other workflows.
TR-02	The OMERO - HPC connection shall be applicable to other workflows.
TR-03	The workload distribution on the HPC system shall be transferable to other workflows.

Table 2.6: Requirements: Transferability

Therefore, it is important that the developed components and design work in comparable system environments.

The requirements listed in this requirement analysis are already categorised into different groups. While the requirements guide the design of the pipeline and its components, Key Performance Indicators can help with decision making, success measurement and prioritising [20]. The identified requirement groups represent KPIs that should be achieved. Table 2.7 describes the KPIs and how the success of the KPI can be achieved.

KPI Name	Description	Measurement Method
Performance	The time the compute-intensive workloads need to compute.	Slurm output logs, profiler, timed tests
Usability	The User Experience for the improved parts of the workflow.	User feedback, user stories
Security	The process, data and system security of the system.	Security analysis
Scalability	The technical scalability of key components.	Pipeline design proposal
Transferability	The transferability of the key components to other similar system environments and use cases.	Pipeline design proposal

Table 2.7: List of KPIs

The Constraint Requirements are integrated into the whole development process and are not a KPI. Each of the other categories of requirements represent a general KPI as a goal. The performance KPI can be measured and timed and a comparison with the existing system can be drawn. Feedback from the target group and user stories help to assess the usability. For the security KPI, an analysis of the threats and vulnerabilities can be made. The success of the KPIs for scalability and transferability depends on the decisions made during the system's design process and the constraints of its operating environment.
The collected requirements and KPIs serve as the foundation for the design and evaluation stages.

Chapter 3

Basics and Related Works

The previous chapters mentioned the RDM system OMERO, some of its system components, and High-Performance Computing. This chapter explains these systems in more detail and provides information that will later be necessary for designing the new pipeline. Section 3.1 gives an introduction to OMERO while Section 3.2 gives an overview about HPC. Section 3.3 is a literature review and and analysis of the related work.

3.1 OMERO

Research Data Management (RDM) is an important part of research in all fields. RDM includes all activities that organise, store, share, document and preserve research data. By adopting a RDM strategy it ensures to improve the scientific quality, reliability and the efficiency of scientific work. A widely accepted guiding principle for RDM is FAIR [21].
The FAIR foundational principles are Findability, Accesibility, Interoperability and Reusability. These foundational principles include 15 principles to allow for machine-actionability and better data management and they guide data producers and publishers in activities concerning digital research [22]. These principles extend to all types of research data and activities. In the context of this thesis, the focus is on bioimaging data.
Bioimaging refers to technologies for viewing biological samples and observing processes and structures of the subject. The technologies included are light microscopy, digital microscopy, electron microscopy, ultrasound, computed tomography scan (CT-scan), magnetic resonance imaging (MRI) and others [23]. Bioimage research data management systems enable RDM strategies for bioimaging research. Over the years some bioimaging research systems have been developed, like Bio-Image Semantic Query User Environment (BisQue) [24], Cytomine [25], OMERO [5] and more. They all share the goal of enabling better support for collaborative research with large image formats.
The focus for this thesis is OMERO. OMERO was first published in 2012 and is now the most

widely used bioimaging RDM system [26]. OMERO is an open-source server software developed by the Open Microscopy Environment (OME) consortium. It aims to reduce vendor lock-in situations because imaging systems often use specialised hardware and software solutions, resulting in proprietary file formats [5]. OMERO supports over 150 file formats and creates new standard file formats, like OME-XML, OME-NGFF and OME-Zarr, to make existing file formats interoperable for different systems. This is mostly done by mapping file informations (e.g. metadata) to a standardised schema, the OME data model [27] [5]. The OMERO system consists of different components and table 3.1 presents the core components, each of which is described in greater detail below.

Name	Description
OMERO.server	Core server application for management of scientific image data.
OMERO.web	Web-based interface for the OMERO.server. Allows users to interact with the image data.
OMERO.insight	Desktop client for OMERO.server. Handles the uploading of image data.
OMERO.cli	Command-line interface for OMERO.server.
OMERO API	Programming language bindings for software development. Provides API to access and manipulate image data.

Table 3.1: OMERO core components

The **OMERO.server** is the core component of OMERO. It is a java-based server application software and consists of multiple systems and features [27]. Figure 3.1 shows a overview of some its parts.

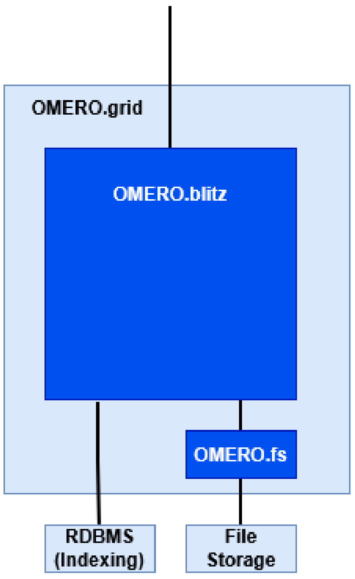
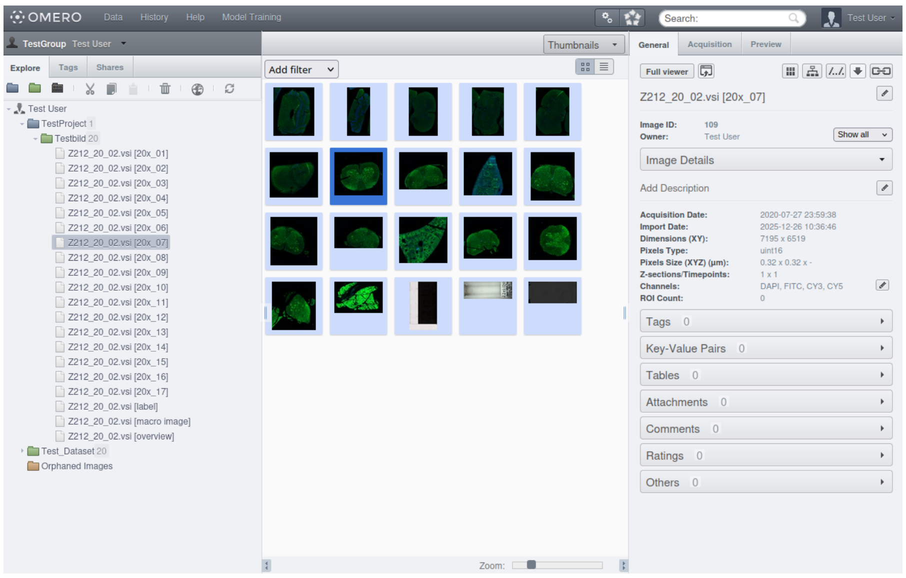


Figure 3.1: OMERO.server architecture

The OMERO.server software is responsible for coordinating the data storage and data management, user management and access control, data import and export and it provides interfaces for other systems. The data storage is split in multiple parts, the image data repository, OMERO.fs, that stores the image data in its original data format, a pixel storage and a database in which the metadata of the image is saved in form of the OME data model and a link to the associated image file. By storing the image in its original file format, it is possible to retain all information and it eliminates the need to duplicate the image when preserving the original image and the image in the OME data model format. The database is a Relational Database Management Software (RDBMS) like PostgreSQL and OMERO supports indexing for faster search queries [5] [27].
OMERO.grid is unifying several components of OMERO. It is used to to monitor and control processes that can be distributed over numerous remote systems. With OMERO.grid it is possible to manage multiple OMERO.server process nodes and storage repositories and manage them as one system [5] [27]. It is based on ZeroC's IceGrid framework, a middleware platform that simplifies the creation and management of distributed applications by deploying a central registry where all server processes are managed and a single daemon tool on each node [27] [28].
The OMERO.blitz component is responsible for providing secure access to data and metadata. It manages user sessions by granting access to users and removing closed and old sessions. OMERO.blitz is used by other OMERO components and it allows for OMERO APIs to connect to the OMERO.server. OMERO's user management allows for setting permissions at different levels to ensure controlled access to sensitive and confidential data [5] [27].

The next core system of OMERO is **OMERO.web**. It allows users to interact with the OMERO.server image data with a browser. OMERO.web uses the Python framework Django to generate Hypertext Markup Language (HTML) and JavaScript Object Notation (JSON) files with data from the OMERO.server. It uses the OMERO.blitz interface to access the data. The main applications of OMERO.web are the webclient for navigating the image data, the webgateway that provides rendered images and JSON data to other applications, and the webadmin tools to manage users, groups and permissions. Additionally, it provides aa interface for OMERO.scripts. These are community developed Python scripts with a user interface. Extensions to the interface can be developed as integrated Django applications. The main view of the interface is split into three parts, as shown in Figure 3.2: the project and file browser on the left side, the thumbnail previews of the images in the center and file-specific information on the right side [5] [27].

Another user interface is **OMERO.insight**. It is a desktop application that runs on the user's system and can connect to the OMERO.server. The interface looks similar to OMERO.web and has a project and file browser, a list with thumbnails of the images, file-specific information and it also has an image inspection window. It differs to OMERO.web because it is one of the systems that allow to import new image data. The import to OMERO.server transfers the original files to the file repository managed by OMERO.fs. An import of files to OMERO.server also



Z212_20_02.vsi [20x_07]

Image ID: 109
Owner: Test User

Acquisition Date: 2020-07-27 23:59:38
Import Date: 2025-12-26 10:36:46
Dimensions (XY): 7195 x 6519
Pixels Type: uint16
Pixels Size (XYZ) (µm): 0.32 x 0.32 x -
Z-sections/Timepoints: 1 x 1
Channels: DAPI, FITC, CY3, CY5
ROI Count: 0

Figure 3.2: OMERO.web user interface

includes the generation of preview images as thumbnails for the OMERO applications, creation of the OME data model using the tool Bio-Formats [29] and saving the metadata in the database. Some image file formats support pyramidal images. These are potentially large images that are hard to load in memory at once. Instead, tiles of the image data are stored at multiple image resolutions in a pyramid-like structure. The original image is downsampled to create a series of smaller images. Each level of the pyramid shows the image at a different level of detail. This means that only parts of the image need to be loaded into memory at a higher resolution. For images that don't include the pyramidal data, OMERO creates them upon import [5] [27].

Another way to connect to the OMERO.server is with the OMERO Command Line Interface (**OMERO.cli**). It is set of Python tools used for system-administration, deployment and for more advanced users of OMERO. It is also the second way to import image data and supports bulk imports for large numbers of files and datasets [5] [27].

For the development of different tools or automation of workflows it is necessary to have the **OMERO API**. Developed for the API are multiple programming language bindings. These include Java, Python, C++ and MATLAB [5] [27].

The components of OMERO allow for external applications to access the image data. Therefore there are plugins for other open-source bioimaging applications that directly connect to OMERO, like the image processing tools Fiji, Napari, ilastik, QuPath and more. Efforts to develop interfaces for various tools are also driven by the OMERO bioimaging community.

3.2 High-Performance Computing

An HPC system is used to solve computationally intensive tasks much faster than a normal computer. This enables researchers, analysts, and engineers to solve more complex problems more time-efficiently.
An HPC system consists of several computing nodes. Each node has at least one Central Processing Unit (CPU), memory and also at least one network interface. The nodes are connected using high-speed, low-latency networks to allow for efficient computations. The interconnects are InfiniBand, high-speed Ethernet or proprietary solutions like Omni-Path [30] [31].
HPC systems can vary greatly in scale, as can be seen in the Top500 [32] list, but the general architecture remains the same. The main components of an HPC system are three types of nodes: the login (or head) nodes, the compute nodes and the storage nodes. Figure 3.3 depicts a typical architecture.

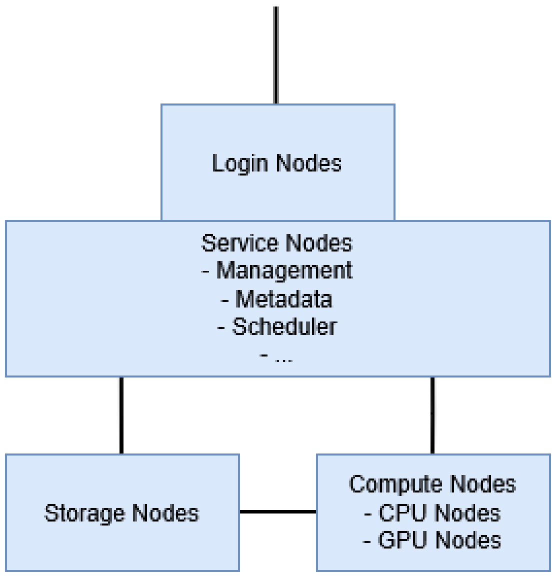


Figure 3.3: HPC architecture

Additionally to the three main node types, there are the service nodes, that are used to manage the system. They are utilised for monitoring, provisioning, scheduling, providing metadata, and more. For the stability of the system, sufficient redundancy of the service functions has to be implemented. The compute nodes provide the processing power for the tasks. They are divided into CPU nodes and Graphics Processing Unit (GPU) (or accelerator) nodes. The CPU nodes are general-purpose processing nodes, while the faster GPU nodes are for specialised tasks [30] [31].
The user of an HPC system usually connects to the login node using software that enables SSH connections. On the login nodes each user has its own environment for files, processes and software modules. It is also possible to group users for collaborative projects to share storage space. The

login node is used to prepare the workflow by organizing and transferring data to and from the HPC system. The compute nodes are not accessed directly. They are controlled by the scheduler software on the service nodes, which manages the access. The scheduler software receives requests for the compute nodes and manages the distribution of tasks according to a set of rules, as shown in Figure 3.4.

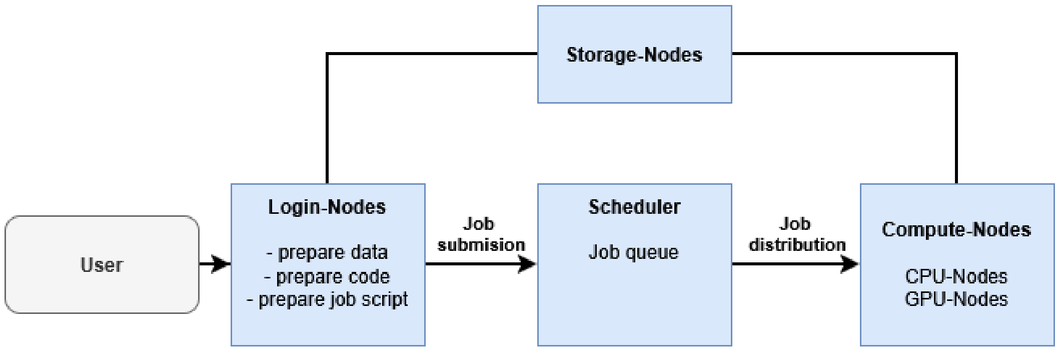


Figure 3.4: HPC scheduling

Nodes with similar characteristics can be grouped into partitions. Partitions are created based on hardware properties, queue policies, job size, or they are used for controlling account permissions. To take advantage of an HPC system, the user must program the tasks (or jobs) accordingly. Each compute node has a fixed number of cores and threads, and the nodes can communicate with each other. The program, which starts with the job script, can be more efficient if it can compute as large a part as possible in parallel. The parallel execution can reduce the wall-clock time if the task and the storage I/O scale properly. Therefore, it is necessary to choose a fitting parallel programming model to distribute the tasks into smaller tasks that can be executed at the same time. Among the the most prominent parallel programming models are distributed-memory models, shared-memory models and GPU and accelerator programming models [33]. Each of these models has a different approach to parallelisation and is suitable for different problem structures.
The most well-known application of the distributed-memory model is the Message Passing Interface (MPI). Its goal is to enable communication and data exchange between independent processes. These processes each have their own memory space and are often located on different HPC compute nodes. MPI supports point-to-point communication as well as collective communication functions that send messages to a group of processes. A challenge of MPI programs is the potential for communication overhead to limit the scalability. This can happen when the number of nodes and processes grows, while the problem size remains fixed. Another challenge is communication latency and data locality, which can slow down the application [34].
OpenMP is an implementation of the shared-memory model. The shared-memory model is used on single nodes with a high processor core count. It uses multithreaded programming and has access to the same memory space for each thread. OpenMP simplifies multithreaded programming by using pragmas that automatically handle thread creation and workload distribution. The scala-

bility of shared-memory applications can be a challenge because they are limited to using a single compute node. Because of this, hybrid models of distributed- and shared-memory programming can be used. These are more complex, but they combine the advantages of both approaches [33].
The GPU and accelerator programming models, as the name suggests, use GPU or General-Purpose Computing on Graphics Processing Units (GPGPU) hardware. These hardware units have a large number of GPU processing cores that are specialised for parallel computing of simple, fixed tasks. The parallelism is on-chip and therefore highly efficient. It is mostly used in the context of the "Single Instruction, Multiple Data" (SIMD) classification, where the same instruction is executed on multiple GPU cores in parallel, but with different data. A challenge of this model can be the transfer of data between the CPU and GPU. They operate at different speeds, which can result in a bottleneck [33].
Examples of the applied model include Compute Unified Device Architecture (CUDA) from NVIDIA, Heterogeneous-Compute Interface for Portability (HIP) from AMD, and Open Accelerators (OpenACC).
Table 3.2 summarises the characteristics of the three approaches.

Characteristic	**Distributed-memory models**	**Shared-memory models**	**GPU and accelerator programming models**
Communication	Message passing	Shared memory	CPU - GPU transfer
Parallelism	Process-level	Thread-level	On-chip
Hardware	Distributed Cluster Nodes	CPU Cores/Threads	GPGPUs
Examples	MPI	OpenMP	CUDA, HIP, OpenACC

Table 3.2: Characteristics: parallel programming models

3.3 Related Work

Advances in bioimaging technologies are generating larger volumes of image data with improved spatial and temporal resolution [4]. This enables new scientific approaches to data analysis, while simultaneously increasing the demand for greater processing power, storage capacity, and data-transfer capabilities. Processing a single data batch can take several days, and the resulting throughput may be insufficient for certain workflows. HPC systems can help address these challenges, and tools that perform parallel data processing are increasingly important [35].
OMERO, as a research data management platform, tries to support interoperability with existing tools by standardising file formats and providing developers with several programming-language bindings for its API. Depending on the requirements, OMERO can be deployed as a full installation on a single server, distributed across multiple servers, or as a simple container for small-scale deployments such as development or test environments. OMERO offers support for integrating

third-party tools to use the data stored in OMERO. These tools include popular bioimaging software such as Fiji, CellProfiler, QuPath, and more [27].
Pierre Pouchin et al. [36] introduce a workflow that uses the open-source image processing package Fiji for ImageJ as the user interface and scripting library, with OMERO serving as the backend data management software. They developed a Java client library for interacting with the OMERO server, a Fiji plugin to run macro programs on batches of microscopy images, and new macro functions to facilitate OMERO interaction for developers. Their goal is to increase the accessibility of image analysis functionality for users of the OMERO platform. The connectivity of the library depends on the Java OMERO API. The plugin retrieves image data from the OMERO server and performs the image analysis on the local workstation. The results can be saved locally or imported into OMERO.
This workflow setup can be extended with the work from Jan Kožusznik et al. [37].They showcase another Fiji plugin that connects to a middleware software based on the HPC-as-a-Service concept. The middleware they use is the High-End Application Execution Middleware (HEAppE). It manages information about submitted jobs and the data flow between the client system and the HPC environment. HEAppE uses SSH connections to the HPC system with password- and key-based credentials, which are secured on the server using the HashiCorp Vault tool [38]. This open-source tool is used to encrypt the user credentials and implement access control and audit logging.
Another adjacent approach is described by Samuel S. Welborn et al. [39]. They utilise the HPC system of the National Energy Research Scientific Computing (NERSC) and the laboratory of the National Center for Electron Microscopy (NCEM) in California. Through a web application called Distiller, the user can issue jobs the the HPC system. The authentication of the user is handled with Open Authorization 2.0 (OAuth). OAuth 2.0 is a token-based authorisation framework in which a resource owner, the user, interacts with a client software system. The client software attempts to access resources on a resource server by redirecting the user to the authorisation server, where the user's identity is verified. The authorisation server then sends a temporary token to the client software, which can use this token to perform the requested actions [40]. Distiller has access to the images of the NCEM laboratory in real time. The images are loaded into memory at the NCEM, and when a job is issued, the image data is transferred directly to the compute nodes' memory using a streaming service that relies on the network socket library Zero Message Queue (ZeroMQ). This avoids writing to storage and overcomes I/O bottlenecks. This approach is specialised for the system environment. The user can also initiate data analysis using NERSC's Jupyter ecosystem running on the HPC system via Distiller [39].

A different approach, more directly linked to OMERO and HPC systems, is the BioImage analysis in OMERO **(BIOMERO)** framework [41] [42]. The BIOMERO framework allows users to start bioimage analysis workflows from the OMERO system and connects to an HPC system to run the computational steps. Its goal is to provide users with a set of predefined bioimage analysis workflows, while also enabling developers to integrate additional containerised workflows for HPC processing. BIOMERO connects to the HPC system via SSH and exports the image data from OMERO, transferring it to the HPC storage nodes. The results are later reimported into OMERO.

3.4 Analysis of Related Work

The related work, particularly the BIOMERO framework, is discussed with respect to the requirements identified in Section 2.3. A gap analysis is conducted by comparing the current state of the related work to the desired state of the system, highlighting its limitations and gaps.

The **architectural** or **constraint** requirements regarding OMERO and an HPC system overlap only with the BIOMERO framework. The other publications focus either on the extended OMERO platform ecosystem or on other bioimaging systems that connect to HPC resources. The connections to the HPC system discussed are primarily SSH connections, while one publication introduces an OAuth connection. BIOMERO makes use of several OMERO components like OMERO.server, OMERO.web, OMERO.scrips and OMERO Django web applications. On the HPC system BIOMERO uses Slurm jobs for workload managing, Singularity/Apptainer for containerised workload environments and Git for additional scripts.

The **security** requirement regarding the secure transfer of sensitive image data is largely fulfilled by using SSH to transfer the data via an encrypted tunnel. The approach of Distiller is not considered because it focuses on memory-to-memory data transfers and performance, and it makes no mention of encryption. BIOMERO and the other publication using SSH store SSH credentials for every user on their external server systems. This introduces a potential security risk to the HPC system and its provider. By storing the SSH details and allowing full user access to the HPC system, an attacker could have access to a number of user accounts. This could cause severe harm to the system and to user data. Possible risks include the misuse of compute resources for malicious purposes (e.g., cryptomining) or intentionally slowing down the system, the theft of sensitive data, especially medical data, privilege escalation by exploiting system vulnerabilities, the modification of data or code, resulting in false outputs, and the installation of backdoors.

All the mentioned approaches aim to improve the **usability** of running image analysis on an HPC system. They incorporate graphical interfaces that users are already familiar with. While Fiji and OMERO use external systems, Distiller also directly supports Jupyter Notebooks on the compute nodes. All interfaces are directly tied to their specific workflow scenarios.

The main goal of connecting to an HPC system is to achieve **performance** improvements. All publications discuss these improvements, and some showcase them on a particular workflow. The performance benchmarks are specific to the system, data, and workflow.

The BIOMERO framework uses the regular OMERO system components and doesnt mention the initial setup of OMERO. A concern regarding **scalability** is that every image is exported as a Zarr file on the OMERO.server before being transferred. This image data conversion could impact server performance when a large number of images need to be processed simultaneously. With image data redundancy, the storage is also temporarily impacted.

BIOMERO allows additional workflows to be added, addressing certain aspects of the **transferability** requirements. The format for these additions is fixed to code on GitHub, a Singularity/Apptainer container, and the development of a custom user interface for OMERO.scripts. This approach adds generality for other workflows, but it can also be challenging to integrate more flexible workflows and existing ones.

Aspect	**BIOMERO**	**BIOMERO requirement fulfillment**	**Thesis contribution**
Architecture / Constraints	OMERO components / SSH / Slurm	Fulfills requirement	Satisfy the constraints
Security	OMERO functionality / SSH Tunnel	Fails requirement	Secure HPC access / Secure data transfer
Usability	OMERO.scripts / OMERO web app	Depends on the implementation	Intuitive user interface
Performance	Improvements based on workflow, system, data	Depends on the implementation	Performance improvement of existing workflow
Scalability	OMERO components	Fails requirement	Scalable design concept
Transferability	OMERO.scripts / Container / GitHub	Depends on the implementation	Transferable design concept

Table 3.3: Gap analysis

Table 3.3 compares the BIOMERO framework with the identified requirement groups and outlines the targeted contributions of this thesis.
BIOMERO is the only related system proposal that satisfies the constraint requirements. No other approach links OMERO directly to HPC resources for image analysis. This may indicate a research gap in a developing field. Most publications assume that the traditional HPC system uses an SSH connection to authorise users. Another approach mentioned is OAuth, which is not yet commonly used in HPC environments. The SSH connection can be a risk factor for HPC systems, especially if a number of SSH credentials are managed on an external system not controlled by the HPC system provider. This risk has to be mitigated. Although usability and performance are emphasized

as primary criteria in the proposed systems, security receives little attention. The usability and performance are strongly linked to the specific workflow and its users, the implementation, and the system environment. The BIOMERO framework allows developers to integrate both new and existing workflows. For general image analysis, it is a suitable solution, however, for multi-step workflows involving different systems, it can be a limiting factor.

Chapter 4

Design

Based on the analysed requirements and the information gained from the previous chapter, a pipeline for an OMERO to HPC system workflow is designed. For this, the architecture of OMERO is discussed in Section 4.1. In Section 4.2, the user interaction and user interface are presented. Section 4.3 explains and evaluates the options for a remote HPC access, and Section 4.4 covers the design decisions regarding the workflow. Section 4.5 concludes the design with a summarising overview of the general architecture of the proposed pipeline.

The goal of designing a pipeline for workflows that utilise OMERO and HPC systems is to derive guidelines and decisions for other, similar workflows operating in comparable system environments. Therefore, the CellDetector workflow introduced in Chapter 2 serves as an example of what a similar workflow could look like and how it could be improved.

A major bottleneck of the current CellDetector workflow is the way other users interact with the HPC computing steps. These users have little to no experience operating on a typical shell interface connected to an HPC system. This entry barrier prevents users from working efficiently and shifts the workload to colleagues who have greater HPC expertise. To address this bottleneck, the most frequent and time-consuming steps should be improved and simplified to make them accessible to more users.

The preprocessing steps described in Section 2.2 are examples of such steps. To make them more accessible, several components are needed. Starting with a GUI that initiates the three functions, sends additional information and connects to OMERO, a connection method is needed to link this interface with the HPC system, in this case, the HPC login nodes. This connection method must be able to connect and authenticate the OMERO user with an HPC user account. From the HPC login node, tasks must be automatically sent to the job scheduler. It should also be possible to start different jobs and workloads without using the shell interface. To analyse the WSIs stored on the OMERO server, they must be made available to the HPC system. This includes the secure transfer and storage on the HPC storage nodes. To improve the efficiency of the analysis, the computing steps should be able to utilise the processing power of the HPC compute nodes. The results should

be made viewable to the user from the OMERO workspace. Furthermore, the designed system should be able to scale with increased user activity and growing storage needs for the bioimage data.
To improve the user experience, the following design shifts the user interaction from a traditional HPC model, where the user connects to and interacts with the system through a shell interface, to a more workflow- or use-case-driven approach, in which the user interacts through a GUI connected to established software that interfaces with the HPC system. The differences between the user interaction with an HPC system are shown in Figure 4.1.

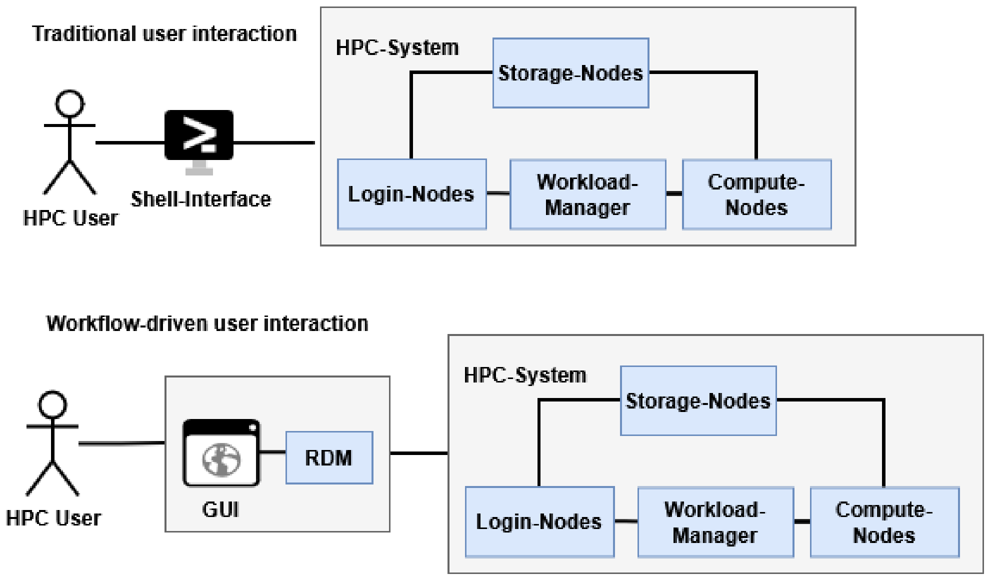


Figure 4.1: HPC user interaction approaches

The traditional interaction approach also supports running software modules with GUIs directly on the compute nodes, if they are supported by the HPC system provider, such as Jupyter Notebooks. Such a module can be run as a batch job that is managed by the scheduler [43].
By changing the way the user interacts with the HPC system, it can also influence user behaviour and productivity. Bradlee Rothwell et al. [44] observe usage patterns after integrating the HPC web platforms Open OnDemand and NICE DCV. Open OnDemand includes HPC features such as drag-and-drop file access, job management, and one-click launching of interactive applications like Jupyter or MATLAB, as well as personalised remote desktops. All of these functionalities are available through a GUI for the HPC user. Their study shows that, after introducing Open OnDemand to their users, the average time between HPC account creation to first job submission decreases. They conclude that this is the result of reduced friction between the user and the HPC system.
The evaluation of the design choices is divided into four parts. Afterwards, an overview of the pipeline's architecture is provided.

1. **Research Data Management System:** Deployment, administration and scalability decisions.
2. **User Interface:** Evaluation of interface options.
3. **Remote HPC Access:** Overview and analysis of different remote HPC access designs.
4. **OMERO Access and Code Optimisations:** OMERO access decisions and use-case specific considerations.
5. **Pipeline Architecture Design Proposal:** Combined view of the architecture and functionality.

The following Sections discuss these different parts and their design in more detail.

4.1 Research Data Management System

The research data management system used in this pipeline is OMERO. The requirements outlined in Section 2.3 specify the constraints for this use case. For a more general architectural design, the deployment of OMERO must also be taken into account. The deployment depends on various factors, such as the existing system environment, costs, security and data policies, available IT expertise, and more. For general workflows, several OMERO components are required. OMERO.server is the central server platform. OMERO.web provides a web-based application and serves as the main interface for OMERO users. OMERO.insight is an optional user client that must be downloaded and configured by the user. OMERO.insight, together with OMERO.cli and the API, is used for data import.
There are multiple deployment methods for an OMERO system. Bare metal, Virtual Machines (VMs), and containers can be used to set up an OMERO environment. Table 4.1 provides a brief comparison of these methods. While the bare metal deployment method offers the best perfor-

Aspect	Bare Metal	Virtual Machine	Container
Performance	High (direct hardware access)	Slightly lower (virtualization layer)	Slightly lower (container overhead)
Scalability	Low (fixed hardware)	High	High (supports orchestration)
Flexibility	Less flexible for moving	Easier to move	Very flexible
Complexity	Higher initial setup (OS/config)	Moderate	Can be automated (requires container knowledge)
Isolation	Low	High	Medium

Table 4.1: Comparison of deployment methods for OMERO

mance, it is not advisable for a flexible and scalable OMERO system architecture because the hardware is fixed and cannot be scaled quickly. For OMERO, it is recommended to use either VMs, in the form of virtual or cloud servers, or custom OMERO containers. Regarding scalability in

the VM approaches, it is easier to allocate or adjust resources for cloud servers than for virtual servers, because the latter are limited by the underlying physical hardware, whereas cloud servers are built on distributed infrastructure. For containerised services, there are two main scaling approaches: horizontal scaling and vertical scaling. Horizontal scaling adds more instances of the same container, while vertical scaling increases the resources of a single container. By default, containers don't retain internal data when removed, so many containerised services are designed to be stateless and scale well horizontally [45]. OMERO.server uses stateful functionality and doesn't scale well horizontally. OMERO.web is designed to be stateless and can scale horizontally. OMERO.server is responsible for user sessions and access to file storage and must be consistent, but it is possible to manage multiple nodes with outsourced worker processes via OMERO.grid.
OMERO is a customisable platform and can add or remove functionalities as needed by its users. This includes OMERO.script and the ability to add new Django applications that have access to OMERO user and image data through the OMERO API. For the virtual server and cloud server approach, the process of adding and removing functionalities is straightforward. For the containerised approach, this means that the container image's definition file must be modified, the image rebuild and the container redeployed. For non-production systems, such as development environments, containers enable fast deployment cycles. Another difference is the rollback capabilities that are common for VMs, such as system and disk state snapshots. Containers don't capture the full system state, and for storage volumes outside the container scope, filesystem snapshots would be needed to achieve similar functionality. From a scalability standpoint, both the virtual/cloud server and the containerised server approaches are viable. The differences lie in how the system must be deployed, modified, and maintained, which also depend on the existing infrastructure as well as the expertise and preferences of the IT team.
Alongside the OMERO core systems, additional features are needed to support a scalable system environment. If multiple nodes, such as OMERO.web nodes, are deployed, the incoming network traffic must be evenly distributed. This can be achieved with a load balancer that ensures no single node receives too much traffic, preventing a decline in performance and availability of the service. OMERO also supports Lightweight Directory Access Protocol (LDAP) authentication. LDAP is used to access and maintain distributed directory information over a network [46]. The directory is a hierarchical, tree-like structure used to store information about users, groups, and other resources. LDAP allows other applications and systems to look up and authenticate users. If the OMERO.server is connected to the network's LDAP service and a new user registered in the LDAP directory tries to log in to OMERO, the server first searches its own user list and then queries the LDAP service with the username. If the username and password exist and match, a new OMERO user is created with the given username and added to a predefined OMERO group. The LDAP service is also queried to check whether a user is no longer registered in the directory, in which case access is denied [27]. LDAP support enables the management of large numbers of users across different systems.
There are few examples of OMERO system architectures public available. One example is dis-

cussed by Anett Jannasch et al. [47] for the Core Facility Cellular Imaging (CFCI) at the Technische Universität Dresden. They use a VM with 16 gigabytes (GB) of memory and initially started with 2 terabytes (TB) of storage, but had to increase it to 20 TB. They utilise a containerised approach with OMERO.server, OMERO.web, and Traefik containers, with Traefik acting as a reverse proxy and load balancer. They authenticate users using an OpenLDAP container that is connected to the institute's LDAP servers.
Another example is from the Image Data Resource (IDR) [48] managed by the University of Dundee and OME. The IDR is a community project that serves as a public repository for image data from scientific studies and currently stores 415 TB of data. The architecture consists of seven cloud servers. One PostgreSQL server is for the central database. One internal server with read and write permissions for OMERO.server and OMERO.web with 16 CPUs and 64 GB memory. Additional IDR has four servers with read-only permissions for OMERO.server and OMERO.web with 8 CPUs and 32 GB memory each. The last component are front-end proxies with Nginx and Haproxy that load balance and cache the web traffic and the access to the OMERO API. These multiple OMERO servers can access the central database in parallel, but only one server has write permissions, while the other public servers have read-only access. Otherwise, it could lead to data inconsistencies.
The evaluation of the deployment methods and the comparison of existing public approaches lead to a general OMERO system architecture, as illustrated in Figure 4.2. OMERO.server should

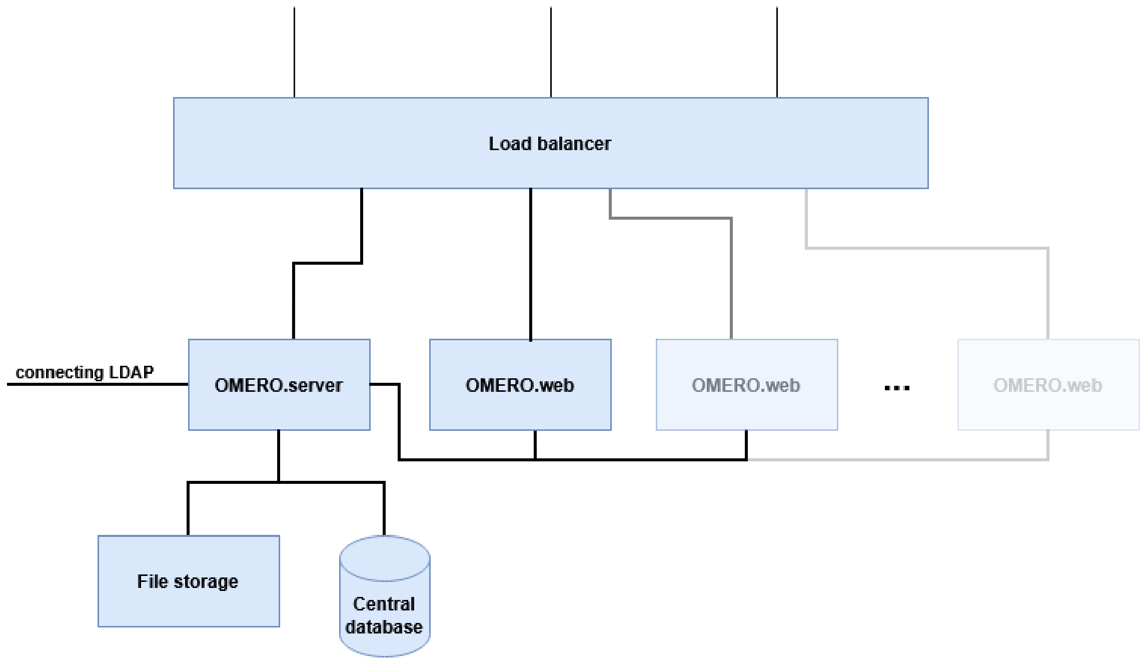


Figure 4.2: General OMERO architecture

be able to scale vertically, whether it is deployed as a VM or a container. OMERO.web can scale horizontally as a VM or a container if traffic or performance demands it. LDAP is optional and

depends on whether other existing systems have already integrated it. The central database and file storage should not reside on the same node, and there should be support for data backup and recovery.
The system architecture of the CellDetector workflow is determined by the requirements in Section 2.3. It uses two separate virtual servers for OMERO.server and OMERO.web, which mitigates some performance concerns and is viable for the current user base. If the user base increases, adjustments would have to be made.

4.2 User Interface

From the user's perspective, the user interface represents the core functionality for working with data stored in OMERO. For the pipeline design, there are three possible ways to integrate a user interface with OMERO in a manner that fits the requirements described in Section 2.3 and one additional option. These will be introduced, discussed and evaluated in the following.

OMERO.scripts
OMERO.scripts are similar to a plugin system for OMERO. It allows users to run automated scripts on images and metadata stored in OMERO. Popular scripts are integrated into the core scripts in newer OMERO versions. The scripts are mainly written in Python, but OMERO also supports additional languages such as Jython and MATLAB. The scripts support tasks such as image analysis, batch processing, data import/export, and metadata management. They can be started from OMERO.insight and OMERO.web. OMERO.blitz enables the scripts to be run on the OMERO.server, and script execution is passed to OMERO.grid, which delegates the node where the processing step is conducted. Each script has its own simple GUI, which is integrated into the interfaces of OMERO.insight and OMERO.web. The GUI is automatically generated from the script. The GUI elements are limited to a few commonly used types, such as text fields, dropdown lists, and checkboxes. The scripts support auto-filling text fields as lists with the selected images in OMERO.web and OMERO.insight. The GUI isn't fully customisable and only generates elements according to the defined variable types required for user input [5] [27].

OMERO.web application
OMERO.web is a Django-based web client for OMERO. It also supports the creation of user-designed custom web applications based on the Django framework. The Django framework uses the Model-View-Controller (MVC) architectural pattern, which separates the user interface, programming logic, and data model layer [49]. These web applications can leverage the core OMERO.web application by extending its existing functionality. The Uniform Resource Locator (URL) of each application is an extension of the OMERO.web address. They use the user's active session to connect to OMERO and load data using the Python API. The web application can be linked to from OMERO.web or integrated into the OMERO.web page layout, such as the center or

right panel [27].
The web applications are fully customisable but need knowledge of the Django framework.

Web application linked in OMERO.web
Similar to user-created OMERO.web applications, it is also possible to create external web applications and link to them from OMERO.web. These can be independent of the Django framework and use a different software stack. To connect to OMERO and access image data and metadata, they must use the OMERO API. They are fully customisable.

Standalone application
A standalone application could be downloaded and used as an external application to connect to OMERO using the OMERO API. It could use any GUI framework and would be fully customisable. However, a standalone application does not fully fulfill the requirement UR-03 in Section 2.3, which states that the GUI must be started from OMERO.web.

Table 4.2 compares the GUI options in several categories. OMERO.scripts and OMERO.web

Aspect	**OMERO.scripts**	**OMERO.web application**	**Linked web application**	**Standalone application**
Type	Server-side script, callable from OMERO.web and OMERO.insight	OMERO.web extension (Django-based)	Separate web application linked to OMERO.web	Independent application
Integration Level	High	High	Medium	Low
User Interface	Auto-generated form for parameters	Fully customisable Django application	Fully customisable web application	Fully customisable standalone application
OMERO Data Access Method	OMERO.web, OMERO API	OMERO.web, OMERO API	OMERO API	OMERO API
Strengths	Directly in GUI, simple implementation	Directly in GUI, flexible	Flexible, can use any web framework	Flexible, can use any framework
Limitations	Limited customisability, few UI elements	Requires Django expertise	No direct OMERO embedding possible	Requirement not fulfilled, Most complex

Table 4.2: Comparison of GUI options

applications are the most integrated options. They are called directly from the OMERO.web interface, which benefits the workflow. For simple workflow steps, OMERO.scripts is the best option. The scripts are simple to implement and provide features to support the user, such

as automatically filling an image ID list with the images selected in OMERO.web. The GUI options for OMERO.scripts are very limited, which restricts its possible use cases. OMERO.web applications are suitable for more complex use cases. They are fully customisable and can be embedded into OMERO.web. Separately linked web applications can be an option for existing workflow steps that should be more integrated into OMERO. They don't have to be reimplemented as a Django framework application, and they can be accessed from OMERO. While independent standalone applications are an option, they don't fulfill the requirements and cannot be integrated into OMERO.web. This would mean that every user would have to download an additional application, which would need to be provided on a file server. This option is more useful for applications that offer a wider range of functionalities, such as Fiji and Napari, and additionally support an OMERO connection.
The preprocessing and prediction steps of the CellDetector workflow can be implemented using OMERO.scripts. These are Python scripts with clearly defined parameters that can be mapped to user inputs in the OMERO.scripts GUI. The cell annotation step can be implemented using OMERO.web applications or by linking the existing Jupyter Voilà application from within OMERO.web. This step needs to display parts of the WSI in different dye channels to the user, which is not possible with the functionality of OMERO.scripts.

4.3 Remote HPC Access

To compute parts of the workflow on an HPC system, it is necessary to create a connection between the OMERO system environment and the HPC system. An OMERO user should be able to start a compute-intensive task from the GUI launched via OMERO.web. This task should schedule one or more jobs on an HPC system without further interaction from the user. This requires a communication interface that works for various workflows. Furthermore, the connection must be secure and, if possible, transferable to other similar HPC system environments. For traditional HPC systems, there are a few options for connecting to the HPC login nodes. These differ in terms of user and system interaction and security levels, and they depend on the policies set by the HPC system provider. Most of the possible solutions are based on SSH connections, as they are widely used and another option is a Representational State Transfer (REST) API.

SSH: Password-based authentication
The first option is the common password authentication. The user has to provide the password to establish an SSH connection. This could be either an additional input field in the GUI or a file stored on the OMERO server. If the authentication succeeds, it remains active until it is closed, for example when the user closes the connection, an idle timeout occurs, or a network-related failure happens. During this connection period, the OMERO system would have full access to the HPC user account. Password-based authentication is the least secure of the SSH connection methods [50].

SSH: Cryptographic key-based authentication
With cryptographic key-based authentication, a public/private key pair is created on the OMERO server. The public key must be stored in the user space of the HPC account or with the HPC provider's authentication service to authenticate the OMERO user's identity. Similar to the password-based authentication method, the connection allows full access to the HPC user account, but the user doesn't have to provide the authentication key as input for each connection [50].

SSH: Certificate key-based authentication
The certificate-based authentication also works with public/private key pairs. Additionally, it provides a Certificate Authority (CA) service that signs the user's public key to create a certificate. The certificate is saved on the OMERO server together with the public/private key pair. During authentication, the CA's public key is used to verify the user's certificate and grant or deny access to the HPC system. The certificate contains information about the user's identity, the public key, the certificate type (user or host), allowed principal names, start and end timestamps, and SSH client–specific options. The timestamps act as a revocation mechanism for SSH certificates. This authentication method does not require the user's public key to be stored in the user space of the HPC account or with the HPC provider's authentication service, because only a valid certificate is needed and a configured CA identity provider. Similar to the password-based and key-based authentication, it grants full HPC user acces [50].

SSH: Forced commands
SSH forced commands are not a specific authentication method. They work as an authorization or SSH session restriction mechanism. The authentication occurs using public/private keys or certificates. The SSH command can invoke remote commands that are executed on the HPC system. The command is chosen by the owner of the keys. Forced commands change this behaviour by retaining control of the commands on the HPC system. By setting a forced command, every successful SSH connection to the HPC system executes the specified command on the HPC node without granting full user-space permissions. The forced command is limited to one command per key. It can be set by the user or enforced by the SSH system administrator. When a forced command is used with certificates, it must be passed to the CA when creating the certificate [50].

REST API connection
In addition to the SSH methods, there are a few approaches that aim to replace the SSH connection with HTTP calls, like HPCSerA [51] and Superfacility API [52]. There the user is authenticated via a client that provides tokens for specific tasks. These tokens are used in REST calls to an API server to request these tasks. The task is then executed on the HPC system by an application that retrieves available tasks from the API server. This method does not grant access to the HPC user

account but requires the necessary system setup.

Table 4.3 shows a comparison of the different remote access options. The SSH password

Option	Limited Access	Transferability	Complexity/Setup
SSH Password	No	Low/Medium	Low
SSH Key-Pair	No	High	Low
SSH Certificate	No	Low/Medium	High
SSH Forced Commands Key-Pair	Yes	High	Medium
SSH Forced Commands Certificate	Yes	Low/Medium	High
REST API	Yes	Low/Medium	High

Table 4.3: Comparison of remote connection options

option is not viable. Passwords are the least secure option because they are prone to poor user practices, such as short character length, password reuse, and lack of randomness. Furthermore, they are not always allowed on HPC systems. In contrast, SSH key pairs are used by many HPC systems. They are more secure because they eliminate poor user practices. They must be stored on the system that will connect to the HPC system. SSH certificates are the most secure of the three SSH authentication methods. They can have a validity period to grant or deny access. The limiting factor of this method is the required system infrastructure, including a CA. None of these three methods are viable for the OMERO-to-HPC pipeline because they grant the accessing system full access to the connected HPC user account. If an attacker captures the authentication credentials, they could directly harm the HPC system. SSH forced commands help address this problem by authorising only a single command per key for an SSH connection. While forced commands used with certificates are more secure, they require a CA setup. This limits their transferability. The REST API approaches do not require an SSH connection. Depending on their implementation, they can be more secure than the SSH options but require more complex system setups. The limitations on the access of HPC user accounts, the transferability and the setup complexity make the SSH forced commands option with public/private key pairs the most promising option for the scope of this thesis.

By choosing the forced commands method with key pairs, HPC user keys must be accessible to the pipeline systems. This means that if the system managing the user keys has a vulnerability, an attacker could gain long-lived authentication credentials. Using forced commands with these keys helps diminish the potential harm. Another option is to prevent the OMERO server from handling the key management and instead use a separate server. This server would run a queue system and register the calls from the OMERO GUI. It could verify the calls and place them in a queue. The queue server would then perform the SSH calls to the HPC system. This would reduce the risk if the OMERO server has a vulnerability and separate their responsibilities.

An example of such a system could be HyperQueue [53], a lightweight system with a built-in queue that can execute commands through its worker nodes. Figure 4.3 illustrates how such a

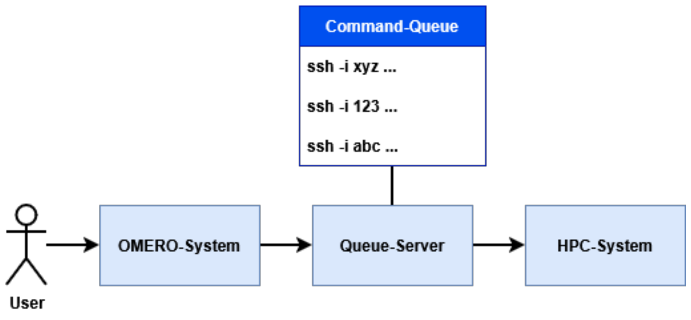


Figure 4.3: Queue-Server

process could look like.

In the CellDetector workflow, the OMERO.server must manage the SSH key pairs because the initial number of users is small and it would need another server that is not considered in the current setup. Instead, an additional application manages the SSH keys. It must be able to create and delete users, manage SSH keys (create, delete, and update), and establish a connection between the keys and the OMERO users.

4.4 OMERO Access and Code Optimisations

Once a connection to the HPC system is established, the workflow can be processed. The CellDetector workflow requires image data from the OMERO.server and reports the (partial) results back to the OMERO.server. BIOMERO retrieves image data by exporting it on the OMERO server and transferring it to the HPC system using Secure Copy (SCP) over a full SSH connection. This approach is incompatible with SSH forced commands. Allowing direct data transfer under forced commands would weaken the restriction, for example, the forced-command file itself could be overwritten, resulting in full access to the user space. This means that, when SSH forced commands are used, the request to retrieve the image data must be initiated from within the HPC system. To initiate this request, the OMERO API is used to connect to the server via one of its language bindings (Python, Java, MATLAB, or C++) using the OMERO server address and the corresponding port. The OMERO API is used within the job scripts. The jobs are executed on the HPC compute nodes, which are usually protected by stricter firewall settings than the HPC login nodes to enhance system security. The compute nodes need to be able to retrieve the image data from a job call. There are multiple options for this, depending on the security restrictions of the HPC system. The following describes three possible options and examines their potential uses.

Direct access
The simplest approach is to connect to the OMERO.server directly over the internet via the given port, using only the OMERO API. This approach is feasible if the HPC security policy permits internet access from compute nodes for all users, or if certain user accounts are authorised to connect to external addresses. This method is not viable if the HPC security policy is more restrictive. It also requires the OMERO.server to have accessible ports. The data transfer encryption depends on the OMERO settings and the chosen port.

SSH port forwarding
Creating an encrypted tunnel and connecting a local port to a remote port is called SSH port forwarding [50]. The SSH client monitors the local port and forwards all activity through the SSH tunnel to the remote server's ports, see Figure 4.4. With this method, the HPC compute node can connect to the OMERO.server's port, and all data transfers would be encrypted over SSH. With this setup, the OMERO.server's ports do not need to be accessible from the internet. SSH port forwarding relies on regular SSH authentication. For public/private key authentication, a key pair must be set up on the HPC system, with the corresponding key authorized on the OMERO server. This SSH connection would give the HPC user access to the OMERO server, which would introduce a security risk. Like the direct access, this method requires a direct internet connection from the HPC compute nodes to the OMERO server. Because of security concerns, this method is suitable only for development or testing environments.

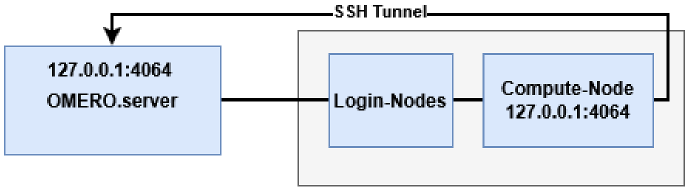


Figure 4.4: SSH port forwarding

SSH reverse port forwarding
Reverse port forwarding works similarly to port forwarding. Instead of listening on the client's port, it listens on the remote port and redirects activity to the client's port [50]. In many HPC systems, it is possible to use SSH to connect from the login node to a compute node allocated to the HPC user. This connection can be used to create interactive sessions, run custom software such as Jupyter, and monitor system resources while a job is running. This SSH connection can be used for reverse SSH port forwarding without creating new keys for authentication. The login node can initiate the reverse port forwarding. This means that the compute node listens on a specified port, and the activity on this port is sent over SSH to the login node. The login node, which is used to prepare and stage jobs and their data, has less restrictive firewall settings. By connecting to the

OMERO server via the login node and forwarding the data to the compute nodes, a connection from the compute nodes to the OMERO server can be established, as shown in Figure 4.5.

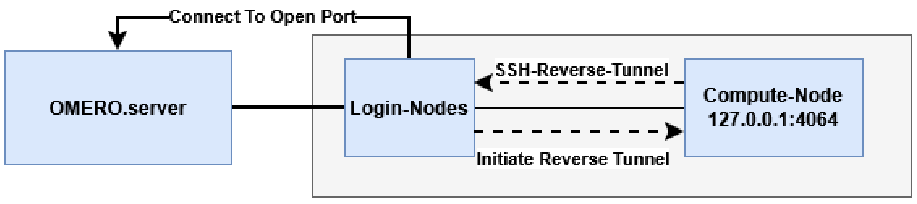


Figure 4.5: SSH reverse port forwarding

Table 4.4 gives a comparison of the three main options.

Aspect	**Direct Access**	**SSH Port Forwarding**	**SSH Reverse Port Forwarding**
SSH Tunnel Initiator	None	HPC compute node	HPC login node
Encrypted Data Transfer	Depending on the OMERO.server configuration	SSH-Tunnel	Depending on the OMERO.server configuration
Setup Complexity	Low	Low/Medium	Medium

Table 4.4: Comparison of OMERO data transfer connection options

Besides the three options introduced, there is also the option to run an application on the login nodes in the user space. This application must be written in a programming language supported by one of the OMERO API language bindings and must have an appropriate runtime environment. It needs to listen for incoming calls from the remote OMERO connection and request the image data from the OMERO.server. To process the image data later, it would need to convert the pixel data and metadata and store them on the storage nodes. While this option is flexible, it also requires the login nodes to run an application and maintain the runtime environment at all times, while potentially processing large amounts of image data simultaneously.
The easiest method to implement is the direct access from the compute node. If this is not possible due to the security policies of the HPC provider, the SSH reverse port forwarding option is the most viable method. With both of these options, the OMERO.server is responsible for the secure transfer of image data and metadata to the HPC system.

Code optimisations

The CellDetector workflow uses the image data and metadata in multiple steps. During preprocessing, they are used for Zarr file conversion, Cellpose image analysis, and eCDF image analysis. While the eCDF analysis already uses the Zarr files, the Cellpose analysis loads the pixel data from the OMERO.server for its computation. This adds an additional data transfer from the external server, which is constrained by the connection's transfer speed. This transfer increases the computation time on the HPC compute node. To reduce this performance bottleneck, the Cellpose step and all subsequent steps should use the Zarr files wherever possible. This concept is described as data locality. Data locality indicates the distance between the data location and the processing location [54]. A shorter distance means better data locality, which results in improved performance and more efficient use of resources [55]. For this, the existing scripts need to be adapted. While the data can be processed on the HPC system, the (partial) results must be made visible to the OMERO user. This requires a connection to the OMERO.server when the results are uploaded.

The most time-intensive computing step is the identification of nuclei in the WSI using Cellpose. The script is written to run on hardware with lower parallelism. While the script benefits from stronger single-core performance on the HPC compute nodes, it doesn't utilize all the resources on such a node. Therefore, the goal is to compute image tiles in parallel with Cellpose using task parallelism, and then combine the partial results.

4.5 General Architecture Proposal

The design decisions from the previous sections form a general architecture for the OMERO-to-HPC pipeline, which is illustrated in Figure 4.6. This pipeline can be used as a guide for implementing

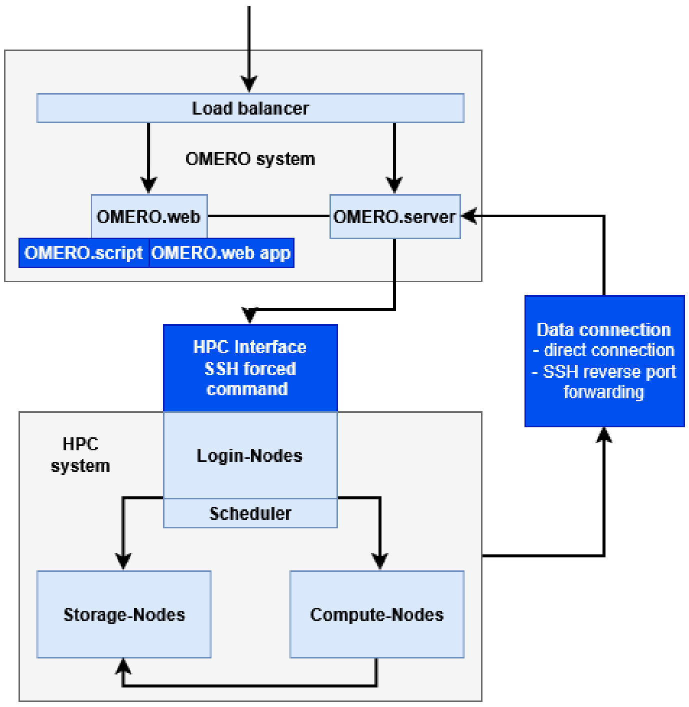


Figure 4.6: Proposed architecture of the OMERO-to-HPC pipeline

workflows similar to the Cellpose workflow. The architecture is scalable and transferable because it uses components commonly found in traditional HPC systems like SSH authentication. The communication interface is created with SSH forced commands, which restrict the actions the owner of the SSH private key can perform. This makes it more secure if an attacker gains access to the key, but it also means that another connection must be formed for data transfer from the OMERO.server. The connection from the HPC system to the OMERO server can be one of two options: a direct connection from the compute node to the OMERO server's open port, or SSH reverse port forwarding with a connection request from the compute node to the login node, and then to the OMERO.server's open port. The option depends on the security policies set by the HPC provider and whether port exceptions are possible for users. The user interface is realised using OMERO's own developer interface options, including OMERO.scripts and OMERO.web applications. This combines the scientific user workflow using OMERO with the options of HPC workflows into one user interface. Each user interface option represents different workflow steps. OMERO.scripts generates a less customizable interface for simple interactions with the stored images, while OMERO.web applications are highly customizable but less integrated and more complex. The HPC computing steps should be able to utilise the parallel hardware and inform the OMERO user of the results from within the OMERO interface.

Chapter 5

Implementation

With the results of the requirement analysis and the design decisions for the proposed pipeline, the Cellpose workflow is optimised. This chapter describes the implementation process and the decisions associated with it. Section 5.1 provides an overview of the key management on the OMERO.server needed for the HPC remote connection. Multiple GUIs are created and discussed in Section 5.2, along with the information they transmit for the next steps. The connection interface and workload distribution are depicted in Section 5.3. Section 5.4 concludes the chapter with the improvements and decisions regarding the existing code and HPC resource usage.

The proposed pipeline in Chapter 4 should guide the implementation of OMERO bioimaging analysis workflows in connection with HPC systems. The CellDetector workflow is constrained to use the existing OMERO.server, OMERO.web, and data repository. This does not limit the current setup, however, scalability must be considered in the future. The implementation consists of two phases: the development phase, in which the changes are developed and tested in an environment similar to the existing system, and the testing phase, in which the changes are applied to the running system to test and evaluate them with live data and users. The existing OMERO.server and OMERO.web run on two separate virtual servers (VMware ESX) with Linux Ubuntu as the operating system (OS). For the development environment, a cloud VM is provided. The specifications are listed in Table 5.1. The OMERO.server, database, repository, and OMERO.web

Type	Cloud VM / Openstack Nova
CPU	4 vCores AMD EPYC Processor x86-64 2.9 GHz base frequency
Memory	16 GB
Storage	80 GB VDisk / vHDD
OS	Ubuntu 22.04.4 LTS x86-64

Table 5.1: Development environment specification

are set up on the cloud VM. The general implementation steps are structured into four sections, which are outlined hereafter.

5.1 Key Management

The remote HPC access is managed with SSH forced commands, as described in Section 4.3. This means that SSH key pairs must be created and managed to access the HPC login nodes. These keys can be stored on the OMERO.server or on a separate server, which can enhance security in the event of vulnerabilities on the OMERO.server. For this project, the keys are stored on the OMERO.server, which is more fitting for the scope of this thesis.
To manage the keys for the users and connect the OMERO accounts to the HPC accounts, a tool is implemented on the OMERO server. The omero-manager is an administration tool written in Bash. Bash is used because it has wide compatibility on Linux and Unix systems, is lightweight, and interacts with popular system tools. The administrative actions are illustrated in Figure 5.1. The

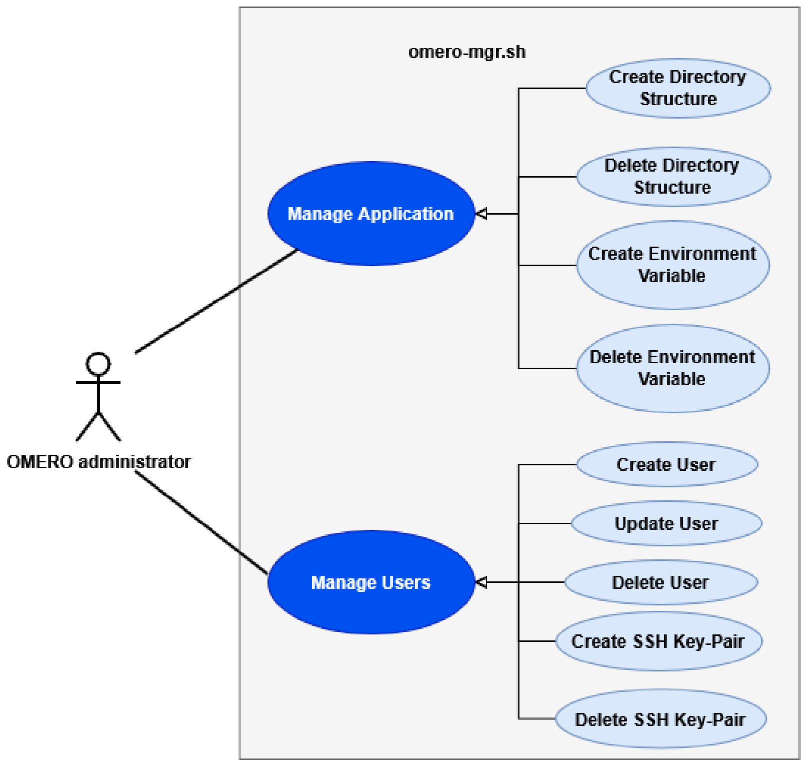


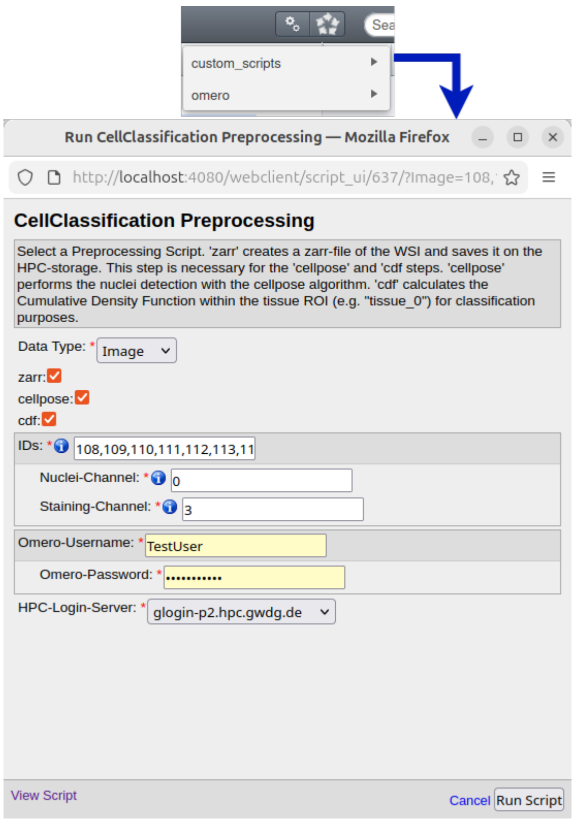
Figure 5.1: Use Case diagram: omero-manager tool

tool is used to create a user space for each OMERO user in the form of directories. Each OMERO user has an SSH key pair that can be recreated (e.g. to adjust its validity period) or deleted. A configuration folder is created for log files. Furthermore, a JSON file is created that links the

OMERO username with the HPC username and the SSH key pair, and adds a timestamp. To access the SSH keys and usernames from the OMERO user interfaces, an environment variable is set that includes the path to the directory created. To assist with administration, a help page is added. The goal of this tool is for it to be quick to learn and use, as user management can be a repetitive task. The access permissions of the files and directories are set automatically. While the path of the set directory is up to the administrator, attention should be paid to ensure that it is not accessible to other users or from the outside. After a new user is created, the public key is used on the HPC system for the SSH forced command setup.

5.2 User Interface

The primary GUI for the preprocessing steps of the CellDetector workflow is the OMERO.scripts interface. The scripts can be written in Python, Jython, and MATLAB and must be uploaded from the OMERO administrator account. The 'CellClassification Preprocessing' script is written in



http://localhost:4080/webclient/script_ui/637/?Image=108,

CellClassification Preprocessing

Select a Preprocessing Script. 'zarr' creates a zarr-file of the WSI and saves it on the HPC-storage. This step is necessary for the 'cellpose' and 'cdf steps. 'cellpose' performs the nuclei detection with the cellpose algorithm. 'cdf' calculates the Cumulative Density Function within the tissue ROI (e.g. "tissue_0") for classification purposes.

Data Type: *	Image
zarr:	✓
cellpose:	✓
cdf:	✓
IDs: *	108,109,110,111,112,113,11
Nuclei-Channel: *	0
Staining-Channel: *	3
Omero-Username: *	TestUser
Omero-Password: *	•••••••••••
HPC-Login-Server: *	glogin-p2.hpc.gwdg.de

Figure 5.2: OMERO.scripts preprocessing user interface

Python. The documentation for OMERO.scripts in Python is the most complete, and Python is popular in scientific workflows, which helps with maintainability if the administrators are already

familiar with the programming language. The scripts use the Python virtual environment of the OMERO.server because they are executed from a server process. If a script needs additional Python libraries, they must be installed in that virtual environment. For the implemented scripts, the Python library Fabric [56] is installed. Similar to Paramiko [57], Fabric enables SSH functionality in Python. Fabric is more high-level than Paramiko and provides the necessary functionalities. The script is written in standard Python, except for an entry point where the GUI elements are defined and the user inputs are read for further actions. The preprocessing script takes the data type, preprocessing step, image IDs, and two dye channels as user input, as necessary information for the analysis steps, see Figure 5.2. Furthermore, it requires the user to choose a login address for the HPC system.
The last input the user must provide is the OMERO login credentials. These credentials are necessary for the connection from the HPC system back to the OMERO.server. Two options are available for authentication: transmitting the OMERO username and password to establish a new connection, or creating a session with the OMERO.script and transmitting the session key. A

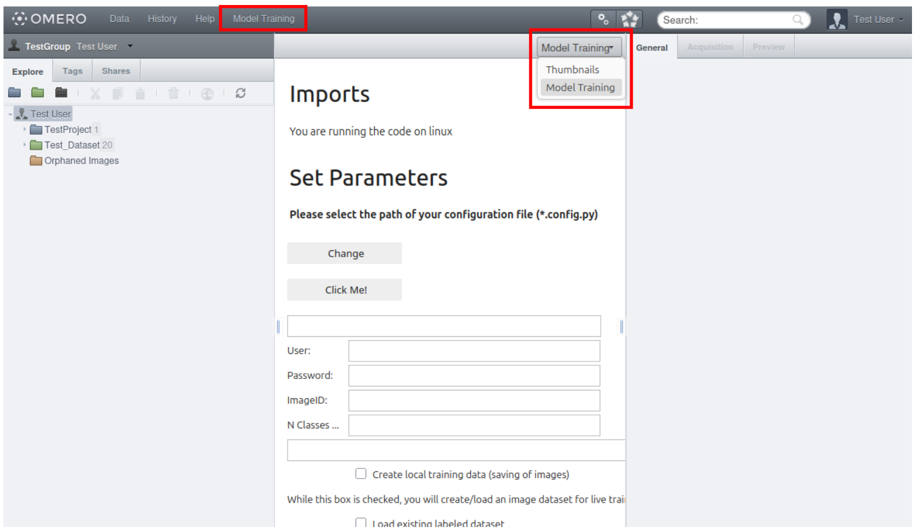
Aspect	**Username/Password**	**Session Key**
Credentials	OMERO username, password Long-lived	Generated character string Short-lived
Security: Credential Complexity	Low/Medium	Medium/High
Security: Session Expiry	Logout, timeout	Logout, adjustable time-out
Security: Vulnerabilities	Brute force	Session hijacking/session replay attack
Flexibility	High	Medium

Table 5.2: Comparison of OMERO authentication options

comparison of the two options is shown in Table 5.2. Session keys are generally more secure due to their length and complexity. They are also shorter-lived, as passwords are seldom changed. The duration of the session key can be specified as an argument when creating the key, and the general length of the OMERO sessions can be adjusted in the server options. A limitation of the session keys is that the session starts when the user interface sends the connection request. If the job cannot start in time, the session may expire. Another limitation is the flexibility of the session keys compared to password authentication. Each OMERO session, regardless of the authentication method, is mapped to an OMERO group, and each image is also mapped to an OMERO group. Every OMERO user can be in multiple groups. When a user logs into OMERO, they are assigned to a default group. To access image data, the user must change the group. With password authentication, each login creates a new session, and each session is mapped to the correct group. If only one session key is sent for multiple jobs, the jobs conflict with each other over the correct group, and a race condition occurs. To achieve the same result as password authentication, the

OMERO.script must send as many session keys as jobs are created to avoid race conditions. This approach has the advantage that sessions can be closed within the jobs, whereas with a single session key the session would have to be left open. Because of its security advantages, session keys are used in the implementation of the OMERO.scripts. The scripts calculate the number of jobs needed and create the same number of session keys. The OMERO API function, which creates detached sessions, requires the OMERO username and password. The user must input the credentials, which are then used in the script on the OMERO.server. These credentials are not transmitted. The input data and session keys create a command string needed for the SSH forced command script. The Fabric library establishes an SSH connection and sends the command string to the HPC login node. The connection port must be specified. By default, the standard SSH port is 22. The preprocessing user interface allows users to select multiple preprocessing steps, images, and dye channels at once, and the images can belong to different groups as long as the user has the correct permissions.
Another implementation of a CellDetector workflow step using OMERO.scripts is the prediction step, depicted in Section 2.2. This script scans the results of the Cellpose step that are uploaded to OMERO as file annotations. It determines how many nuclei candidates were found and calculates how many jobs should be created to parallelise the processing. This information is sent as a command string to the SSH forced command on the HPC system. This interface also allows for multiple images to be selected at once.

As an alternative to OMERO.scripts, the OMERO.web applications are another way to implement workflow steps. OMERO.web and these applications are Django-based web apps. They can be accessed through their own URL paths in the OMERO.web application. In the web



Imports

You are running the code on linux

Set Parameters

Please select the path of your configuration file (*.config.py)

Change

Click Me!

User:

Password:

ImageID:

N Classes ...

☐ Create local training data (saving of images)

While this box is checked, you will create/load an image dataset for live trai

☐ Load existing labeled dataset

Figure 5.3: OMERO.web application integration

interface, they can either be linked from the menu or integrated as web client plugins. As a web client plugin, they can replace or add to existing panels of OMERO.web.
For the CellDetector workflow, the annotation of the cells is integrated using the OMERO.web app feature. As described in Section 2.2, the existing workflow uses Jupyter Voilà as a web application to connect to OMERO and prepare the data for cell annotation. It uses the OMERO Python API to authenticate the user, preselect cells from the results of previous workflow steps, and load the image data. For the integration with OMERO.web, there is the option to recreate the existing functionality with a Django app or to integrate the existing web application. The Django app can be more tightly integrated with OMERO, which uses the Python API to load data and render it into HTML pages using Django templates. The functionality would remain the same, but the OMERO administrator would no longer be able to maintain the application as easily as the Jupyter notebook. For this reason, the integration of the existing web application is the most viable option.
The web application is integrated in two ways, see Figure 5.3. The external address is linked in the top-left menu of OMERO.web and opens in a new browser window. This option does not provide further integration with OMERO. The other integration is a Django web application that uses the web client plugin functionality. The Django application is added to the center panel of OMERO.web and can be accessed via a dropdown menu. The Django application itself uses an iframe to access the existing external address. This enables tighter integration with OMERO functionality because the session of the OMERO.web user can be used. Information from the running session can be used in the external Jupyter Voilà application via the postMessage API. The postMessage API is a browser API that enables message exchange between different windows, tabs, iframes, and workers. The custom Django application reads the OMERO session details and sends them with the postMessage function to the iframe of the external Jupyter Voilà web application. The Jupyter notebook adds a JavaScript event listener that accepts messages from the specified origin address. The message includes the session key of the existing OMERO.web session. For the cell annotation application, this skips the authentication with an OMERO username and password. The postMessage API is supported by nearly all current browsers, including Chrome, Edge, Firefox, Opera, Safari, and more [58]. By default, the Jupyter Voilà application may block embedding in iframes. To allow the OMERO.web application to access the web application, the content security policy of the underlying Tornado web server must be adjusted, as shown in Listing 5.1. This example command starts the web application and allows it to be embedded from any domain. For a safer configuration, the frame ancestor should be set to the domain of the OMERO.web server. This setup allows for tighter integration while keeping the external web application maintainable and largely separate.

```
voila --Voila.tornado_settings='{"headers":{"Content-Security-Policy":"frame-ancestors
    self *" }}' cell_annotation.ipynb
```

Listing 5.1: Content security policy setting for Voilà

5.3 Remote HPC Access and Task Distribution

The OMERO.script user interfaces create a connection to the chosen HPC login node using the Python library Fabric. On the HPC system, the HPC user or an HPC administrator must add the SSH forced command and the generated SSH public key. By default, this is done in the user space of the HPC user under `~/.ssh/authorized_keys` or the `sshd_config` file. The file `authorized_keys` is used for public key authentication. For normal SSH key authentication, it is sufficient to specify the key encryption type, the public key, and an optional comment that labels the key for readability. For SSH forced commands, the command is written before the key specifications, as shown in Listing 5.2.

```
command="nohup script_selector.sh </dev/null >script_selector.log 2>&1 &" ssh-ed25519
    AAAA.... user@example
```

Listing 5.2: SSH forced command setup

The `nohup` command is used to detach the execution of the application from the SSH connection. Without `nohup`, the OMERO user can accidentally terminate the application by logging out of OMERO.web, which ends the SSH connection and the entire execution. The `script_selector.sh` script is the application that the forced command allows to be executed over the SSH connection and it is illustrated in Figure 5.4. No other commands can be executed.

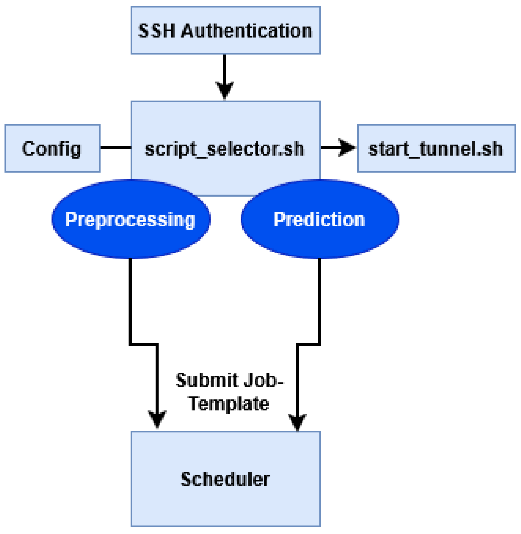


Figure 5.4: Overview forced command

The forced command applications do not work with traditional input arguments, instead, an environment variable called `SSH_ORIGINAL_COMMAND` is set for the active session. The size that the environment variable can reserve for the session is also the limit of the command string. On a Linux system, this size is controlled by `ARG_MAX`, which limits the total size of arguments and environment variables combined. The `script_selector` checks the `SSH_ORIGINAL_COMMAND` for inputs, selects the chosen workflow, and prepares the data. An additional configuration file is used for directory paths, variables, and to declare the connection type from the HPC system back to the OMERO.server. The `script_selector` reads the connection type, submits the job to the scheduler, and, if needed, starts the application for the backward connection. The SSH forced command mechanism is used to limit the attack vector if the SSH key pair is compromised. Therefore, it is important that the forced command application does not allow user inputs to be evaluated or expanded, as this could result in code injection and unwanted command execution.

If the compute nodes are behind a firewall that blocks outgoing connections, the alternative is to initiate SSH reverse port forwarding from the login nodes to the scheduled compute nodes. This option must be supported in the `sshd_config` configuration, with options like `AuthorizedKeysFile`, which sets the directory for the `authorized_keys` files in the user space, and `AllowTcpForwarding`. The `script_selector` has two approaches that function similarly: communicate the node name from the running job and initiate the reverse tunnel, as illustrated in Figure 5.5. The difference is how they communicate. One approach uses two setup

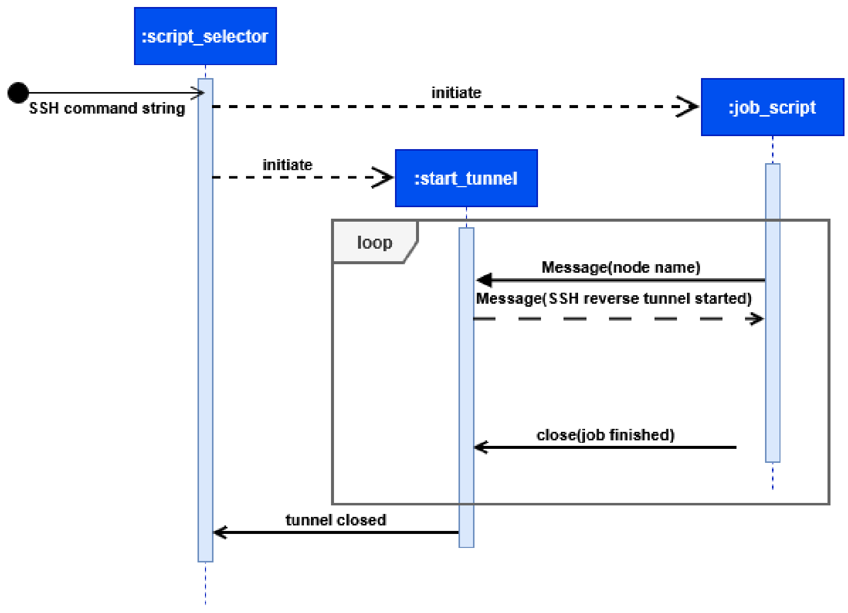


Figure 5.5: Sequence diagram: SSH reverse tunnel

files, and each process must use the `flock` command to acquire the lock on the file in order to gain the right to edit it. Otherwise, race conditions occur when multiple jobs are running in parallel and performing write operations on the files. The other approach creates new files for each communication step. This prevents race conditions but may potentially create many temporary files. On HPC systems that have user limits on inodes, that behaviour may be undesirable. Therefore, the locking approach is the most viable option. SSH reverse

5.4 Code Optimisations

To improve the general performance of the CellDetector workflow, the existing codebase must be adapted for better use of the HPC system's resources. The biggest bottleneck of the existing code is the execution of the nuclei detection algorithm Cellpose. Two potential bottlenecks are identified: the transfer of pixel data from the OMERO.server to the compute node, and the processing of Cellpose without efficiently utilising the compute node.

For the first bottleneck, the existing code is adapted to decouple the OMERO connectivity as much as possible. The only connection is made when the results are stored on the HPC storage and simultaneously uploaded to the OMERO.server as a file attachment to the analysed image file. With these changes, the Zarr file created in the first step of preprocessing can be used for the pixel data, and the metadata of the images is stored separately in a JSON file. This improves data locality by moving the image data closer to the processing nodes, thereby potentially enabling better performance.

The next bottleneck concerns the more efficient utilisation of HPC resources. A main goal of the existing code is general-purpose deployment on a variety of systems, including workstations, laptops, and HPC systems. In its current state, it can potentially use multiple cores by default on workstations and laptops, depending on the system environment. On HPC systems, the usage of resources often has to be implicitly declared and integrated into the code. For many image analysis steps, including Cellpose, data parallelism is the most appropriate parallelism paradigm. The image data is split into separate tiles, and the Cellpose nuclei detection algorithm can be applied to them in parallel. The existing code is written in Python. Python is widely used in scientific workflows and is the most well-documented language regarding the OMERO API. For the Cellpose step, Python multiprocessing is used with its shared-memory parallelisation model. The application itself uses the fork-join model for the parallelisation, see Figure 5.6. The Cellpose step is compute bound in the execution of the nuclei detection.

Another change to the codebase is the implementation of Zarr v3, as the existing code uses Zarr v2. Zarr, as a file format, uses multidimensional arrays that are divided into chunks. These chunks are stored as separate files. Depending on the chunk size specified by the developer and the image size, a single image in the Zarr format can consist of thousands of files or even more. While this structure contributes to the advantages of the Zarr file format, it can also be detrimental

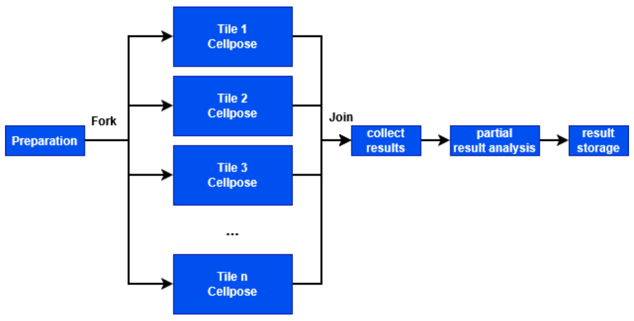


Figure 5.6: Fork-Join parallelisation Cellpose

in HPC environments. On HPC systems that impose user limits on inodes, this restricts the number of Zarr files that can be generated and stored, because every file, directory, and symbolic link is counted as one inode.
Another characteristic of HPC systems is that the performance of processing steps can suffer when there are too many I/O calls, especially in combination with metadata servers. For Zarr files with very small chunk sizes, this could pose an issue.
By using Zarr v3, these issues are addressed through its sharding functionality. Sharding groups chunks and combines them into a single unit, called a shard. The chunks within a shard are still accessible individually, as in previous Zarr versions, but the number of files can now be reduced. The chunks within the files can be read individually, but write operations are performed on the entire shard [16]. Through this functionality, both inode usage and performance issues are improved.

Chapter 6

Evaluation

Based on the changes to the CellDetector workflow that follow the design choices made for the general OMERO-to-HPC pipeline, this chapter provides an evaluation. By connecting OMERO to an HPC system and identifying performance bottlenecks, Section 6.1 evaluates the results of the changes. The usability of the new interfaces is the focus of Section 6.2. Section 6.3 illustrates the security aspects of the pipeline and the associated risks to the systems. Section 6.4 discusses the aspects of scalability and transferability of the system.

The general pipeline design from Chapter 4 and the groups of requirements identified in Section 2.3 form the basis for evaluating the implemented system. The KPIs defined in Section 2.3 determine whether the system's objectives are fulfilled and which shortcomings still exist. The following sections evaluate the functional and non-functional objectives.

6.1 Performance

To fulfill the performance KPI defined in Section 2.3, the runtime of compute-intensive workloads must be improved. The biggest performance bottleneck in the processing steps of the CellDetector workflow is the nuclei detection with Cellpose. Section 5.4 describes the changes that were made. To measure the impact of these changes, tests are conducted. From the provided set of research data, three representative images are chosen. They are chosen based on their file size, as this most strongly influences the workload. The image information is shown in Table 6.1. The smallest image, the median-sized image, and the largest image in the data set are chosen. The tests are conducted

Name	small image	median image	large image
Resolution	4753 x 5167	7336 x 7468	19284 x 27437
Storage size	100 MB	221 MB	2157 MB

Table 6.1: Image information for the performance test

on the GWDG HPC system, on the Emmy P2 CPU partition standard96, with 2 × Sapphire Rapids 8468 processors, 96 cores, and 514,000 MB of memory. All tests are repeated at least 10 times.

Data locality

To assess performance improvements from the changes to data locality, both the existing codebase and the updated version were tested. Since the existing codebase runs using only one core, both versions are tested under the same scenario. The results are illustrated in Figure 6.1. The results show an improvement for all images. The improvements in percentages are

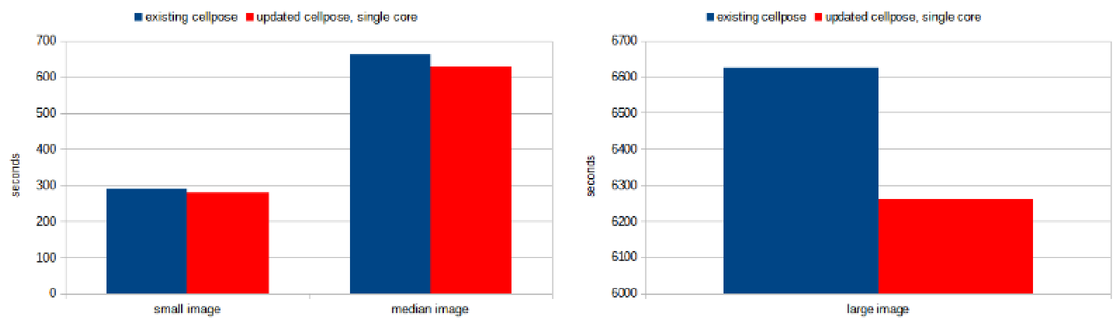


Figure 6.1: Performance tests: Data locality

3.491% (small image), 5.51% (median image) and 5.495% (large image). Since the file sizes of the two smaller images are only 100 MB and 221 MB, the time needed for connection setup and transmission is relatively small. For batches of larger images, the improvements are significant.

Parallelisation

The next set of tests measures the changes to parallelisation with Python using multiprocessing. For each set of tests, the number of cores was increased (1, 2, 4, 8, 16, 32, 64, 128, 192) until the maximum of 192 cores on the specified HPC compute node partition. The results of these tests are shown in Figure 6.2. The graphs show a constant line for one to two cores. To prevent

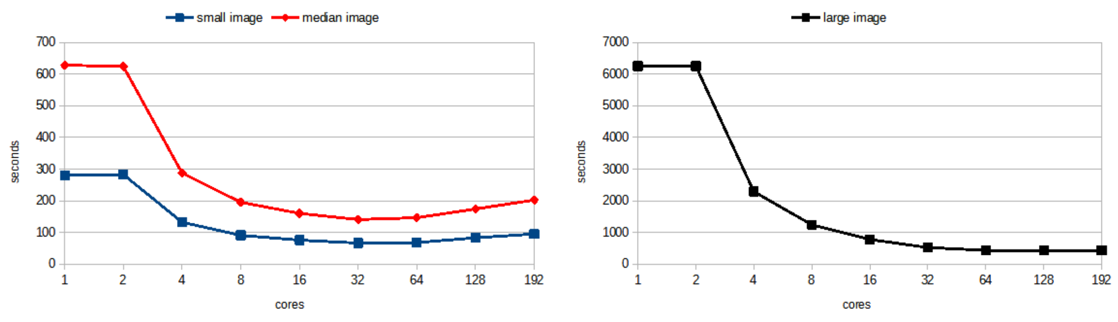


Figure 6.2: Performance tests: Total time parallelisation

slowing down the application when all cores are in use, one core is always reserved. All workload sizes show a significant improvement in performance times. While the large image shows improvements even with over 64 cores, the performance of the two smaller images worsens with more than 64 cores. This is due to the reduced performance gains from increased parallelisation and the added overhead, such as process creation and synchronisation.

Based on the test results, the speedup from the parallelisation changes can be calculated. The results are shown in Figure 6.3. The large image shows the most significant speedup in

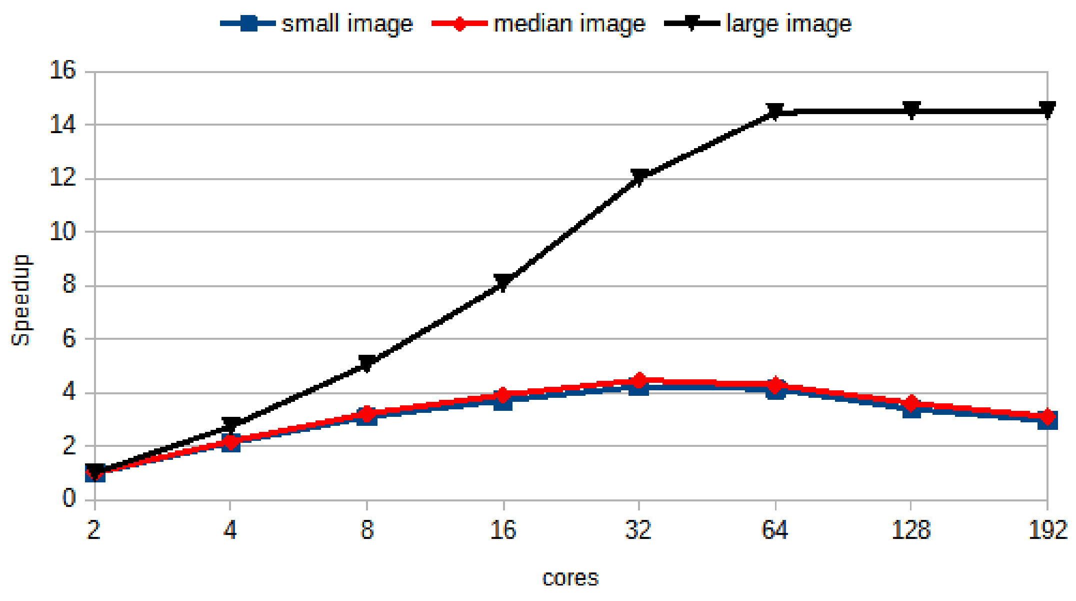


Figure 6.3: Performance tests: Speedup

performance, while the smaller images have a speedup of around 4.2 to 4.5 before losing efficiency with more cores. This is, once again, the result of system overhead. Even for the larger image workload, the speedup levels off at around 14.46 for 64 cores. Adding more cores results in minimal to non-existent improvements.

To show the effect of adding more cores in relation to the speedup, the efficiency decay is calculated. The results are presented in Figure 6.4. The efficiency decay shows that performance gains decrease, and after a certain point, adding more cores no longer improves performance.

The performance testing results show that both performance-related issues with the Cellpose processing step are improved by the changes detailed in Section 5.4. In particular, parallelization improves the performance of all workloads, with larger images gaining the most

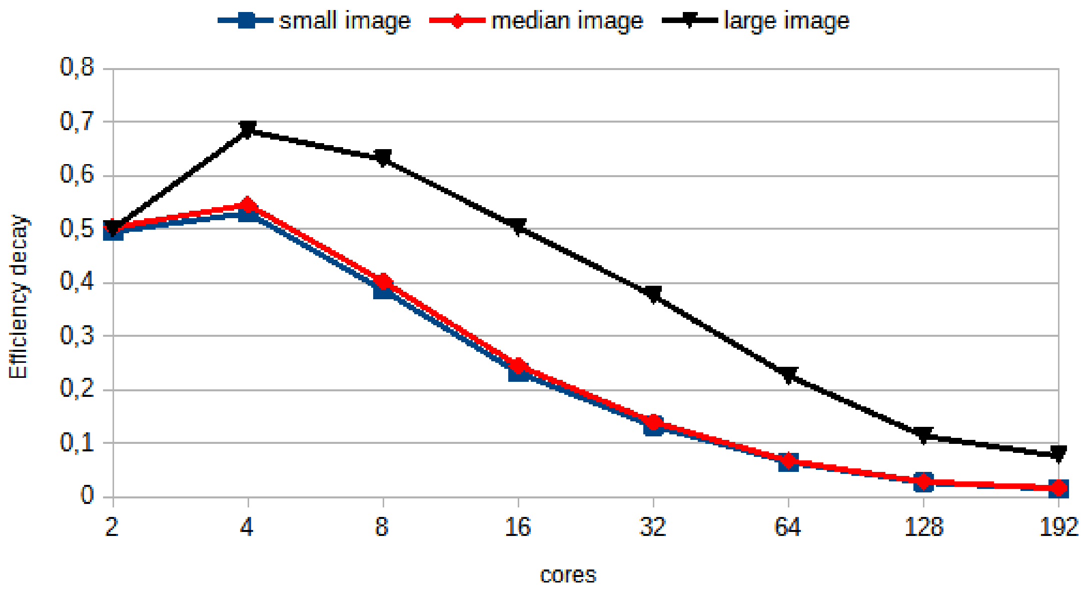


Figure 6.4: Performance tests: Efficiency decay

performance. If the number of cores is not chosen dynamically, it should be based either on the larger workloads, as most processing time is spent on them, or on the most frequently used workloads.

6.2 Usability

A main goal of the OMERO-to-HPC pipeline is to enable more researchers to use workflows that require HPC systems for more efficient research, in combination with OMERO as an RDM platform. One of the challenges of the existing system environment is the potential barrier to entry of HPC systems for new users. That is why, in Section 2.3, one group of identified requirements and the following KPI are about the usability of the workflow. To assess if the designed pipeline in Chapter 4 and the implemented interfaces in Section 5.2 address these issues, the usability of the system must be examined.
Usability can be defined as the "extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" [59].
The term usability is a latent construct. This means that there is no direct way to measure a value that implies the usability of a system. It is possible to derive, indirectly from other observations, how the latent construct usability is affected. In the literature, there are different usability models that try to identify the dimensions that affect usability. The International Organization for

Standardization (ISO) 9241-11 [59] standard defines the dimensions, as shown in Table 5. While

Dimension	Description
Effectiveness	Measured through metrics that focus on the accuracy and completion of tasks.
Efficiency	Efficiency refers to the resources required to achieve a specified goal. These resources are typically measured in terms of time and effort.
Satisfaction	Satisfaction is described through the attributes of comfort and acceptability. They refer to the complexity, ease of use, and learnability of the system.

Table 6.2: ISO 9241-11 Usability dimensions

ISO defines three dimensions, effectiveness, efficiency, and satisfaction, there are other usability models with slightly different dimensions. Another approach is proposed by Jakob Nielsen [60], who defines five attributes in his usability model: learnability, efficiency, memorability, errors, and satisfaction. While learnability, efficiency, and satisfaction overlap in parts with the ISO model, memorability and errors are new. Memorability describes how easy the system is to remember, allowing a casual user to always return to it without relearning everything. The error attribute adds to ISO's effectiveness and accuracy the ability to recover from errors. For the usability testing of the pipeline, the ISO dimensions are used and adapted.

To capture the dimensions of effectiveness, efficiency, and satisfaction, user stories are created that describe the functionalities of the implemented software from the perspective of its users. Additionally, acceptance criteria are defined that are tied to the attributes that determine if a user story is complete. The completion serves as an indicator of whether the attribute is satisfied and the usability is improved.
The user stories are listed in Table 6.3. They aim to map each usability dimension to a few metrics that can be measured in simple user tests. The focus of the usability tests is the OMERO.scripts interface because it provides the HPC connectivity implemented in this thesis. For the research members, the time improvements to user actions are especially important. With the existing system, every image ID had to be written to an external list.
The tests of the user stories were completed successfully. The biggest time investment is the initial setup for new users on an HPC account. A usability concern is the lack of feedback for the HPC jobs once they are started, as the current setup has no functionality for querying the HPC job status. The successful testing of the acceptance criteria for the user stories indicates a usability improvement over the existing system. Where the existing system has a larger entry barrier, the new system is more accessible.

Name	User Story	Acceptance Criteria
Story-1	As a researcher, I want to learn how to launch the CellDetector preprocessing steps after a short introduction, so that I can quickly start processing my data with minimal errors.	- Time needed for introduction $\leq$ 15 min - Number of failed attemps $\leq$ 5 - Time to first successful HPC job creation after introduction $\leq$ 10 min Dimension: **Satisfaction**
Story-2	As a researcher, I want to quickly process large numbers of images using the CellDetector workflow, so that I can analyze them in a shorter time.	- Number of clicks needed to select all images and start the HPC jobs $\leq$ 25 - Time needed for HPC job creation $\leq$ 3 min Dimension: **Efficiency**
Story-3	As a researcher, I want to access the results of the image preprocessing steps, so that I can review and analyze the associated data.	- Successful completion rate of HPC jobs $\geq$ 95% - (uploaded file annotation) / (expected file annotations) $\geq$ 95% Dimension: **Effectiveness**

Table 6.3: Usability: User Stories

6.3 Security

The proposed and implemented pipeline connects two different system environments: the OMERO system and an HPC system. Both systems have different user management and access control mechanisms and store sensitive and confidential data related to patient health and/or research. Furthermore, it is possible that the OMERO system and the HPC system are managed by different providers with different security policies. For these reasons, it is essential to provide a secure approach to bridge the systems and connect their functionalities. For the OMERO system, this mostly concerns the security of data transfer, including encryption, as well as access to the OMERO.server with its primarily password-based authentication. On the HPC system side, this also concerns the authentication of its users, as well as malicious access and misuse of its resources. In the proposed pipeline, both sides must use the authentication mechanism of the other side to run the intended workflow. Based on the security requirements identified in Section 2.3 and the design and implementation from Chapters 4 and 5, this section discusses possible security concerns of the pipeline.

Remote HPC Access

The HPC system authentication in the pipeline is based on SSH key pairs for authenticating the HPC user. The required SSH key pair is stored on the OMERO.server. This means that the HPC user grants the OMERO server the authority to connect to the HPC system and the personal user space. If a malicious attacker gains access to the OMERO.server, for example, through a software or system vulnerability, this would give them attack options for the HPC system and its data. To limit this attack vector, the proposed pipeline uses SSH forced commands. The HPC user limits the access rights of the SSH key by allowing only a single command or application to be callable. The accessible application must never trust the inputs coming from the SSH connection, as it is possible to inject code through them. With code injection, it is possible to run commands or even overwrite the `authorized_keys` file, gaining full access to the HPC user space. By never evaluating or expanding the inputs, the risk of such an attack can be mitigated.

OMERO.server Access

The pipeline design proposes a connection from the HPC system to the OMERO.server to access its data and transmit it for further processing. OMERO uses passwords as its primary authentication mechanism. This means that the potentially long-lived password would have to be sent to the HPC system. To avoid this, a short-lived authentication mechanism would be more secure. Bearer tokens would be such a mechanism. They are short-lived, stateless tokens that can be sent in the authorization header of an HTTP request. OMERO currently does not support such token functionality. Instead, it uses sessions for all user activities. The corresponding session keys are stateful but are also shorter-lived than passwords. Therefore, it is important not to risk session hijacking by exposing the session keys. The session keys are possibly transmitted during three actions: the SSH connection from the OMERO.server to the SSH forced command application on the HPC system, the connection from the HPC system back to the OMERO server, and with the custom OMERO web application, where the session key is transmitted with the webpage data. The first connection uses the encrypted SSH tunnel, the second uses Secure Sockets Layer (SSL) for authentication by default, and the third is transmitted via HTTPS. This means that every transmission of the session keys is encrypted.

In addition to secure authentication, the security of the image data transfer is also important. OMERO uses TCP ports 4063 and 4064 to connect to the OMERO.server. Port 4063 is unsecured, while port 4064 uses SSL. The default OMERO configuration is to use SSL only for authentication and allow all other communication to be unencrypted in order to speed up image loading. By changing the OMERO server configuration or setting `secure=True` in the Python API call, it is possible to always encrypt the communication. This prevents malicious users on the network from spying on the OMERO network traffic.

User Operating Errors

The user must correctly set up the SSH connection and the scripts folder for the pipeline to work. The correct implementation of the SSH forced command is especially important. The scripts include a bash script that takes the public key as input and writes the SSH forced command to `authorized_keys` to minimise small mistakes. The rest of the directory and applications are preconfigured but allow users to change paths and job scripts.

Taking these security concerns into consideration and minimising the risks strengthens the security of the entire pipeline, and with it, both connecting systems.

6.4 Scalability and Transferability

By making the existing workflow more accessible to more users, the pipeline must be scalable to support more users and data. The pipeline also needs to be transferable to other workflows and similar systems. Based on the requirements identified in Section 2.3, the general pipeline architecture was designed in Chapter 4. This design is the foundation for a scalable deployment. By virtualising or containerising the OMERO components, it becomes easier to scale parts in response to increasing numbers of users and data. Offloading the computational steps to the HPC keeps the OMERO server responsive and improves the workflow. The constraint requirements mean that no changes are made to the underlying system, and only new functionalities are added.

To make the pipeline transferable, Bash is chosen for all non-computing step applications. Bash is widely used in Linux and Unix system environments and supports many additional tools. A similar decision was made to use SSH forced commands with key pairs. SSH key authentication remains a popular authentication mechanism for HPC systems. The `script-selector` application allows for the integration and support of new processing steps and workflows. The design decision to choose OMERO.scripts allows for the quick development of new user interfaces for additional HPC computing steps.
The OMERO-to-HPC pipeline is a scalable and transferable approach to connect the OMERO RDM with HPC system resources.

Chapter 7

Conclusion and Future Work

The goal of this thesis is to design a pipeline that connects the workflows of researchers performing microscopy image analysis using the RDM system OMERO with an HPC system and its compute resources. Alongside the pipeline proposal, the CellDetector workflow is adapted, and the OMERO-to-HPC pipeline is implemented to demonstrate and test the design decisions.

To do that, Chapter 2 explains the use case and illustrates the existing workflow and its architecture. Afterwards, a requirements analysis is conducted to identify the requirements and KPIs. In addition to the constraint requirements, five groups of requirements are identified: performance, usability, security, scalability, and transferability. These groups create the basis for the KPIs. From there, Chapter 3 gives an overview of OMERO and HPC systems, and the related works are identified and analysed. The analysis identifies a lack of research in this specific area, and the few existing approaches have, among other things, issues regarding a secure connection to an HPC system. The following chapters design, implement, and evaluate the proposed components of the pipeline.

To achieve the stated goal, several research questions were outlined in Section 1.1 to address the identified challenges. Each question represents a part of the proposed pipeline.

> *Q1: What are the options for improving the performance of analysing and processing microscope images by utilising HPC resources?*

Q1 represents the HPC computing component of the workflow. In Section 4.4, two workflow-specific performance bottlenecks are identified: data transfer of the images and insufficient parallelisation of the analysis steps. Section 5.4 illustrates a solution to these issues with the CellDetector workflow, and Section 6.1 presents the performance impact of these changes. The results are better utilisation of the compute nodes and a significant performance gain in the time-intensive processing steps.

Q2: How can the operation of the HPC resources in this workflow be made more user-friendly?

Q2 includes the usability of the system and the ability to connect to the HPC system. The user interface options are evaluated in Section 4.2, where the decision is made to utilise OMERO.scripts and OMERO.web applications, as they serve different use cases. Section 5.2 illustrates their integration, and Section 6.2 analyses how the usability can be evaluated.

Q3: What potential security issues could arise, and how could they be addressed or minimised?

Q3 combines all security-related issues of the presented pipeline approach. These include the HPC remote access, OMERO.server access, and potential user operating errors. A key component of the pipeline is the use of SSH forced commands to limit the HPC access of the OMERO system. This creates a secure connection interface between both environments. The potential security issues are evaluated in Section 6.3.

Q4: What aspects need to be considered as the amount of data, number of users, and future workflows grow?

The pipeline design in Chapter 4 addresses the scalability and transferability goals by suggesting a scalable deployment of OMERO with VMs and containers and introducing the SSH forced command script, which can be extended to include other workflows. The choice of the tools used makes the pipeline transferable to other similar environments.

These contributions create a pipeline design that can be implemented to enable OMERO users to create HPC-supported workflows without having to learn the traditional utilisation paradigm of HPC systems.

Future Work

For future work, the functionality of the system could be extended to provide OMERO users with more feedback about the running jobs on the HPC. This would require another user interface and could be implemented using OMERO.web applications. It would provide the user with more information about whether the job requests are handled correctly, or if a job was canceled and needs to be restarted. It could also provide additional information about the HPC storage space used by the workflow.

To improve the security of the SSH key pairs, a separate server could be implemented that stores the keys securely and handles all HPC connection requests with a queue system. The OMERO.server would then only send commands to the separate server.

Bibliography

[1] K. Batko and A. Ślęzak, "The use of big data analytics in healthcare," *Journal of Big Data*, vol. 9, no. 1, pp. 1–24, 2022.

[2] S. Gathu, "High-performance computing and big data: Emerging trends in advanced computing systems for data-intensive applications," *Journal of Advanced Computing Systems*, vol. 4, no. 8, pp. 22–35, 2024.

[3] S. Gesing, "Science gateways in hpc: Usability meets efficiency and effectiveness," in *Modeling and Simulation in HPC and Cloud Systems*. Springer, 2018, pp. 73–86.

[4] D. R. Julian, A. Bahramy, M. Neal, T. M. Pearce, and J. Kofler, "Current advancements in digital neuropathology and machine learning for the study of neurodegenerative diseases," *The American Journal of Pathology*, 2025.

[5] C. Allan, J.-M. Burel, J. Moore, C. Blackburn, M. Linkert, S. Loynton, D. MacDonald, W. J. Moore, C. Neves, A. Patterson *et al.*, "Omero: flexible, model-driven data management for experimental biology," *Nature methods*, vol. 9, no. 3, pp. 245–253, 2012.

[6] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of management information systems*, vol. 24, no. 3, pp. 45–77, 2007.

[7] R. S. Weinstein, A. R. Graham, L. C. Richter, G. P. Barker, E. A. Krupinski, A. M. Lopez, K. A. Erps, A. K. Bhattacharyya, Y. Yagi, and J. R. Gilbertson, "Overview of telepathology, virtual microscopy, and whole slide imaging: prospects for the future," *Human pathology*, vol. 40, no. 8, pp. 1057–1069, 2009.

[8] X. Li, C. Li, M. M. Rahaman, H. Sun, X. Li, J. Wu, Y. Yao, and M. Grzegorzek, "A comprehensive review of computer-aided whole-slide image analysis: from datasets to feature extraction, segmentation, classification and detection approaches," *Artificial Intelligence Review*, vol. 55, no. 6, pp. 4809–4878, 2022.

[9] N. Kumar, R. Gupta, and S. Gupta, "Whole slide imaging (wsi) in pathology: current perspectives and future directions," *Journal of digital imaging*, vol. 33, no. 4, pp. 1034–1040, 2020.

[10] N. Kanwal, F. Pérez Bueno, A. Schmidt, R. Molina Soriano *et al.*, "The devil is in the details: Whole slide image acquisition and processing for artifacts detection, color variation, and data augmentation: A review," *IEEE Access"*, vol. 10, pp. 58 821–58 844, 2022.

[11] O. Sertel, J. Kong, H. Shimada, U. V. Catalyurek, J. H. Saltz, and M. N. Gurcan, "Computer-aided prognosis of neuroblastoma on whole-slide images: Classification of stromal development," *Pattern recognition*, vol. 42, no. 6, pp. 1093–1103, 2009.

[12] N. Dimitriou, O. Arandjelović, and P. D. Caie, "Deep learning for whole slide image analysis: an overview," *Frontiers in medicine*, vol. 6, pp. 264–271, 2019.

[13] J. Franz, "Celldetectorpreprocessing," 2025, accessed: 2025-12-31. [Online]. Available: https://github.com/J-Franz/CellDetectorPreprocessing

[14] "What is airflow?" 2025, accessed: 2025-12-31. [Online]. Available: https://airflow.apache.org/docs/apache-airflow/stable/index.html

[15] SchedMD, "Slurm workload manager - documentation," 2025, accessed: 2025-12-31. [Online]. Available: https://slurm.schedmd.com/documentation.html

[16] "Zarr python - documentation," 2025, accessed: 2025-12-31. [Online]. Available: https://zarr.readthedocs.io/en/stable/

[17] C. Stringer, T. Wang, M. Michaelos, and M. Pachitariu, "Cellpose: a generalist algorithm for cellular segmentation," *Nature methods*, vol. 18, no. 1, pp. 100–106, 2021.

[18] "Jupyter - documentation," 2025, accessed: 2025-12-31. [Online]. Available: https://docs.jupyter.org/en/latest/

[19] C. Pacheco and I. Garcia, "A systematic literature review of stakeholder identification methods in requirements elicitation," *Journal of Systems and Software*, vol. 85, no. 9, pp. 2171–2181, 2012.

[20] I. Setiawan and H. H. Purba, "A systematic literature review of key performance indicators (kpis) implementation," *Journal of Industrial Engineering & Management Research*, vol. 1, no. 3, pp. 200–208, 2020.

[21] C. Schmidt, T. Boissonnet, J. Dohle, K. Bernhardt, E. Ferrando-May, T. Wernet, R. Nitschke, S. Kunis, and S. Weidtkamp-Peters, "A practical guide to bioimaging research data management in core facilities," *Journal of microscopy*, vol. 294, no. 3, pp. 350–371, 2024.

[22] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne *et al.*, "The fair guiding principles for scientific data management and stewardship," *Scientific data*, vol. 3, no. 1, pp. 1–9, 2016.

[23] H. S. Lahoti, S. D. Jogdand, and H. Lahoti, "Bioimaging: evolution, significance, and deficit," *Cureus*, vol. 14, no. 9, pp. 1–8, 2022.

[24] K. Kvilekval, D. Fedorov, B. Obara, A. Singh, and B. Manjunath, "Bisque: a platform for bioimage analysis and management," *Bioinformatics*, vol. 26, no. 4, pp. 544–552, 2010.

[25] R. Marée, L. Rollus, B. Stévens, R. Hoyoux, G. Louppe, R. Vandaele, J.-M. Begon, P. Kainz, P. Geurts, and L. Wehenkel, "Collaborative analysis of multi-gigapixel imaging data using cytomine," *Bioinformatics*, vol. 32, no. 9, pp. 1395–1401, 2016.

[26] C. Schmidt, J. Hanne, J. Moore, C. Meesters, E. Ferrando-May, S. Weidtkamp-Peters *et al.*, "Research data management for bioimaging: the 2021 nfdi4bioimage community survey," *F1000Research*, vol. 11, pp. 638–655, 2022.

[27] O. M. Environment, "Omero documentation," 2025, accessed: 2025-12-31. [Online]. Available: https://omero.readthedocs.io

[28] ZeroC, "Icegrid," 2025, accessed: 2025-12-31. [Online]. Available: https://doc.zeroc.com/ice/3.7/ice-services/icegrid

[29] OME, "Bio-formats - documentation," 2025, accessed: 2025-12-31. [Online]. Available: https://bio-formats.readthedocs.io/en/stable/about/index.html

[30] S. Koleini, B. Pahlevanzadeh, and I. Shiraz, "Enhancing high-performance computing (hpc) security: A comprehensive review of detection and protection strategies," *Journal of Distributed Computing and Systems (JDCS)*, vol. 6, no. 12, pp. 12–24, 2024.

[31] C. Engelmann, S. L. Scott, C. Leangsuksun, and X. B. He, "Symmetric active/active high availability for high-performance computing system services." *J. Comput.*, vol. 1, no. 8, pp. 43–54, 2006.

[32] Top500, "Top500," 2025, accessed: 2025-12-31. [Online]. Available: https://top500.org/

[33] M. U. Ashraf, F. Fouz, and F. A. Eassa, "Empirical analysis of hpc using different programming models." *International Journal of Modern Education & Computer Science*, vol. 8, no. 6, pp. 27–34, 2016.

[34] I. Laguna, R. Marshall, K. Mohror, M. Ruefenacht, A. Skjellum, and N. Sultana, "A large-scale study of mpi usage in open-source hpc applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–14.

[35] A. R. Rao, G. A. Cecchi, and M. Magnasco, "High performance computing environment for multidimensional image analysis," *BMC Cell Biology*, vol. 8, no. Suppl 1, pp. 1–9, 2007.

[36] P. Pouchin, R. Zoghlami, R. Valarcher, M. Delannoy, M. Carvalho, C. Belle, M. Mongy, S. Desset, and F. Brau, "Easing batch image processing from omero: a new toolbox for imagej," *F1000Research*, vol. 11, pp. 392–402, 2022.

[37] J. Kožusznik, P. Bainar, J. Klímová, M. Krumnikl, P. Moravec, V. Svatoň, and P. Tomančák, "Spim workflow manager for hpc," *Bioinformatics*, vol. 35, no. 19, pp. 3875–3876, 2019.

[38] T. U. of Ostrava, “Heappe middleware documentation,” 2025, accessed: 2025-12-31. [Online]. Available: https://heappe.it4i.cz/docs/index.html

[39] S. S. Welborn, C. Harris, S. M. Ribet, G. Varnavides, C. Ophus, B. Enders, and P. Ercius, “Streaming large-scale microscopy data to a supercomputing facility,” *Microscopy and Microanalysis*, vol. 31, no. 1–9, p. ozae109, 2025.

[40] E. Ferry, J. O Raw, and K. Curran, “Security evaluation of the oauth 2.0 framework,” *Information & Computer Security*, vol. 23, no. 1, pp. 73–101, 2015.

[41] T. T. Luik, R. Rosas-Bertolini, E. A. Reits, R. A. Hoebe, and P. M. Krawczyk, “Biomero: A scalable and extensible image analysis framework,” *Patterns*, vol. 5, no. 8, 2024.

[42] T. T. Luik, J. de Folter, R. Rosas-Bertolini, E. A. Reits, R. A. Hoebe, and P. M. Krawczyk, “Biomero 2.0: end-to-end fair infrastructure for bioimaging data import, analysis, and provenance,” *arXiv preprint arXiv:2511.13611*, 2025.

[43] A. Reuther, N. Brown, W. Arndt, J. Blaschke, C. Boehme, A. Chazapis, B. Enders, R. Henschel, J. Kunkel, and M. Martinasso, “Interactive and urgent hpc: challenges and opportunities,” *arXiv preprint arXiv:2401.14550*, pp. 1–10, 2024.

[44] B. Rothwell, M. Sgambati, G. Evans, B. Biggs, and M. Anderson, “Quantifying the impact of advanced web platforms on high performance computing usage,” in *Practice and Experience in Advanced Research Computing 2022: Revolutionary: Computing, Connections, You*, 2022, pp. 1–8.

[45] H. Schmidt, Z. Rejiba, R. Eidenbenz, and K.-T. Förster, “Transparent fault tolerance for stateful applications in kubernetes with checkpoint/restore,” in *2023 42nd International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2023, pp. 129–139.

[46] N. Dimitrijević, N. Zdravković, M. Bogdanović, and A. Mesterovic, “Advanced security mechanisms in the spring framework: Jwt, oauth, ldap and keycloak,” in *Proceedings of the 14th International Conference on Business Information Security (BISEC 2023)*, 2024, pp. 64–70.

[47] A. Jannasch, S. Tulok, C. W. Okafornta, T. Kugel, M. Bortolomeazzi, T. Boissonnet, C. Schmidt, A. Vogelsang, C. Dittfeld, S.-M. Tugtekin *et al.*, “Setting up an institutional omero environment for bioimage data: Perspectives from both facility staff and users,” *Journal of microscopy*, vol. 297, no. 1, pp. 105–119, 2025.

[48] U. of Dundee and OME, “Image data resource,” 2025, accessed: 2025-12-31. [Online]. Available: https://idr.openmicroscopy.org/about/deployment.html

[49] S. Chen, S. Ahmmed, K. Lal, and C. Deming, “Django web development framework: Powering the modern web,” *American Journal of Trade and Policy*, vol. 7, no. 3, pp. 99–106, 2020.

[50] D. J. Barrett and R. E. Silverman, *SSH, the Secure Shell: the definitive guide*. " O’Reilly Media, Inc.", 2001.

[51] M. H. Biniaz, S. Bingert, C. Köhler, H. Nolte, and J. Kunkel, "Secure authorization for restful hpc access," in *INFOCOMP 2022, The Twelfth International Conference on Advanced Communications and Computation*, 2022, pp. 12–17.

[52] B. Enders, D. Bard, C. Snavely, L. Gerhardt, J. Lee, B. Totzke, K. Antypas, S. Byna, R. Cheema, S. Cholia *et al.*, "Cross-facility science with the superfacility project at lbnl," in *2020 IEEE/ACM 2nd Annual Workshop on Extreme-scale Experiment-in-the-Loop Computing (XLOOP)*. IEEE, 2020, pp. 1–7.

[53] J. Beránek, A. Böhm, G. Palermo, J. Martinovič, and B. Jansík, "Hyperqueue: Efficient and ergonomic task graphs on hpc clusters," *SoftwareX*, vol. 27, p. 101814, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711024001857

[54] Z. Guo, G. Fox, and M. Zhou, "Investigation of data locality in mapreduce," in *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. IEEE, 2012, pp. 419–426.

[55] D. Unat, A. Dubey, T. Hoefler, J. Shalf, M. Abraham, M. Bianco, B. L. Chamberlain, R. Cledat, H. C. Edwards, H. Finkel *et al.*, "Trends in data locality abstractions for hpc systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 10, pp. 3007–3020, 2017.

[56] J. Forcier, "Fabric documentation," 2025, accessed: 2025-12-31. [Online]. Available: https://www.fabfile.org/

[57] ——, "Paramiko documentation," 2025, accessed: 2025-12-31. [Online]. Available: https://www.paramiko.org/

[58] Mozilla, "Window: postmessage methode," 2025, accessed: 2025-12-31. [Online]. Available: https://developer.mozilla.org/de/docs/Web/API/Window/postMessage

[59] ISO, "Ergonomics of human-system interaction — part 11: Usability: Definitions and concepts," 2025, accessed: 2025-12-31. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en

[60] J. Nielsen, *Usability Engineering*. Morgan Kaufmann, 1994.