Infrastructure for digital medicine and AI enhanced pathology

Dissertation zur Erlangung des

Doktorgrades der Naturwissenschaften (Dr. rer. nat.)

der Fakultät für Biologie und Vorklinische Medizin

der Universität Regensburg

vorgelegt von

Michael Huttner

aus

Regensburg

im Jahr

2024

Das Promotionsgesuch wurde eingereicht am:

29.02.2024

Die Arbeit wurde angeleitet von:

Prof. Dr. Rainer Spang

Unterschrift:

Michael Huttner

Abstract

Modern cancer research relies on a vast array of different technologies and data sources. Many of them use next-generation sequencing (NGS) techniques, with diverse application domains such as genomics, transcriptomics, epigenomics and metagenomics. These processes create data on the molecular level, and enables researchers to analyze genetic mutations, gene expression patterns, genome methylation, protein-DNA interactions and much more. With falling costs, research projects are including more and more NGS experiments, creating massive amounts of data that requires secure storage, processing and analysis.

The advent and impressive progress of powerful AI models in recent years has created another important source of data from imaging. Pathologists analyze histopathological images to identify cellular abnormalities, tissue structures and patterns indicative of certain cancer subtypes. The pathologist's diagnosis will often decide the cancer therapy used, and getting it correct is a matter of life and death. The amount of data, the complexity of it, and the regulatory requirements render data-management a central task of utmost importance for the success of research projects.

This thesis describes the development and utilization of digital infrastructure for both NGS sequencing and AI-based pathology image analysis. We modularize this infrastructure through the use of software containers as basic building blocks. We build mostly independent services and manage them automatically by using kubernetes container orchestration. We created infrastructure for securely storing and sharing large datasets and for managing NGS metadata. We show how this is applies for NGS projects in cancer research. In the same way we built services for managing and acquiring pathology image data and for running AI workloads on these pathology images. This is the foundation for applying deep neural networks to diagnose lymphoma subtypes on a dataset of slides images from 157 patients. Our software focuses on usability by researchers and leverages modern web technologies for both creating a graphical user interface for users and enabling computational access through JSON APIs.

Our projects are available open-source, and they are in active use for further research, both in NGS sequencing as part of the TRR305 project on metastatic organ colonization in cancer, and for pathology image analysis as part of the FDLP project on federated learning on lymphoma pathology data.

Zusammenfassung

Die moderne Krebsforschung stützt sich auf ein breites Spektrum von unterschiedlichen Technologien und Datenquellen. Viele von ihnen nutzen Sequenzierungstechniken der nächsten Generation (NGS) mit verschiedenen Anwendungsbereichen wie Genomik, Transkriptomik, Epigenomik und Metagenomik. Diese Verfahren erzeugen Daten auf der molekularen Ebene, und ermöglichen den Forscher*innen die Analyse von Genexpression, Gen Mutationen, Protein-DNA-Interaktionen, DNA Methylierung und vieles mehr. Mit kontinuierlich sinkenden Kosten, werden in Forschungsprojekten immer mehr NGS-Experimente durchgeführt, die große Datenmengen erzeugen. Daten, die eine sichere Speicherung, Verarbeitung und Analyse erfordern.

Das Aufleben und die enormen Fortschritte leistungsstarker KI-Modelle in den letzten Jahren haben eine weitere wichtige Datenquelle geschaffen: die Bildanalyse. Patholog*innen analysieren histopathologische Bilder zur Identifizierung von zellulären Anomalien, Gewebestrukturen, und um Muster zu erkennen, die auf bestimmte Krebs Subtypen hinweisen. Die Diagnose der Pathologin oder des Pathologen entscheidet oft über die angewandte Krebstherapie, und die richtige Diagnose ist lebenswichtig für Patient*innen. Die Menge der Daten, ihre Komplexität und die strengen gesetzlichen Anforderungen machen das Datenmanagement zu einer Aufgabe von großer und zentraler Bedeutung für den Erfolg von Forschungsprojekten.

Diese Arbeit konzentriert sich auf zwei miteinander verbundene Interessensgebiete: NGS-Sequenzierung und KI-Histopathologie. Zunächst wird die Infrastruktur beschrieben, die zur Unterstützung der Arbeit in diesen beiden Bereichen geschaffen wurde. Das wichtigste Ergebnis daraus ist ein hochverfügbarer Kubernetes-Cluster. Er bietet uns eine verteilte Plattform für containerisierte Dienste und Pipelines.

Für die NGS-Sequenzierung haben wir Software entwickelt die alle Sequenzierungsdaten, die wir erhalten, mit starker Kryptografie verschlüsselt. Dies ist für genetische Daten von Menschen erforderlich, aber wir wenden Verschlüsselung standardmäßig auf alle Sequenzierungsdaten an. Die Daten werden nur symmetrisch verschlüsselt gespeichert, und die symmetrischen Schlüssel werden dann mit Public-Key-Kryptographie gesichert. Dieses hybride Kryptosystem hat große Sicherheitsvorteile, denn der Datenzugriff erfordert sowohl den Zugriff auf die Daten als auch den Besitz der richtigen Schlüssel. Selbst wenn die Server kompromittiert werden sollten, bleiben die Daten sicher. Mehrere potenzielle Risiken wurden berücksichtigt. Wir haben verschiedene Wiederherstellungsoptionen dokumentiert und implementiert, z. B. für den Fall eines Schlüsselverlusts durch die Benutzer*innen. Die wichtigste Innovation dieser Software ist die Verwendung der Web Crypto API im Webbrowser des Benutzers für die gesamte Kryptografie. Die Nutzer*innen müssen keine Soft-

ware installieren, und die kryptografischen Algorithmen sind in den Browser integriert, wo sie ständig überprüft werden und die Hersteller der Browser ihre Sicherheit gewährleisten. Bei unserem Entwurf müssen sich die Nutzer*innen zwar nicht aktiv mit der Kryptografie befassen, aber sie müssen trotzdem mit den Daten richtig umgehen und ihre Schlüssel sichern. Wir haben die lokale Erzeugung von Kryptoschlüsseln implementiert und können sie auf gedruckten QR-Codes speichern, die dem mentalen Modell für physische Schlüssel sehr ähnlich sind.

In Verbindung mit den NGS-Daten sammeln wir die relevanten Metadaten zentral in einem Webtool. Die Benutzeroberfläche ähnelt einem einfachen Spreadsheet, das den meisten Nutzer*innen vertraut ist. Doch im Vergleich zu einer einfachen Excel-Tabelle werden die Metadaten durch die zentrale Infrastruktur über den gesamten Lebenszyklus der Daten hinweg synchron gehalten. Wir haben Metadatenvorlagen auf der Grundlage bestehender Metadatenstandards, z. B. denen des European Genome-Phenom Archive (EGA), entwickelt. Die Metadatenstandards werden auf benutzerfreundliche Weise, durch Dropdowns und sofortige, visuelle Datenvalidierung durchgesetzt. Die Daten werden als vollständige Zeitleiste der Ereignisse gespeichert, so dass sie selbst bei versehentlichem Überschreiben wiederhergestellt werden können. Da sowohl die Metadaten als auch die Sequenzierungsdaten über eine API zugänglich sind, können Pipelines die Daten automatisch lesen und schreiben.

Für die KI-Pathologie beschreiben wir zunächst die gemeinsame Infrastruktur, die wir für unsere KI-Projekte geschaffen haben. Dazu gehören die Datenaufnahme, die Konvertierung von Pathologie Objektträger Scans in ein offenes Format, die Visualisierung der digitalen Bilder und der schnelle Lesezugriff für Deep Learning. Anschließend stellen wir unsere Arbeit an der KI-Diagnose von Lymphomen anhand eines Datensatzes von 628 digitalisierten Objektträgern von 157 Patient*innen. Wir verwenden Transfer Learning, um tiefe neuronale Netze auf einer großen Menge von Patches, Bilder von kleinen Teilen des Objektträgers, zu trainieren. Diese trainierten Netze werden dann auf Testdaten angewendet, um Diagnosekarten zu erzeugen. Das sind lokal annotierte Objektträgerbilder, die die KI-Diagnose für verschiedene Regionen der Objektträger anzeigen. Diese Karten können von Patholog*innen ausgewertet werden, und wir können alle lokalen Diagnosen zu einer Patient*innen Diagnose zusammenfassen. Für den Datensatz von 157 Patient*innen erzielen wir eine gute Quote von 60% richtig klassifizierter Patches und waren in der Lage, alle Patienten korrekt zu diagnostizieren. Leider ließ sich diese Leistung nicht auf einen unabhängigen Datensatz übertragen, und es ist weitere Arbeit erforderlich, um ein Modell zu erstellen, das auf verschiedenen Datensätzen gut performt.

Summary

Modern cancer research relies on a vast array of different technologies and data sources. Many of them use next-generation sequencing (NGS) techniques, with diverse application domains such as genomics, transcriptomics, epigenomics and metagenomics. These processes create data on the molecular level, and enables researchers to analyze genetic mutations, gene expression patterns, genome methylation, protein-DNA interactions and much more. With falling costs, research projects are including more and more NGS experiments, creating massive amounts of data that requires secure storage, processing and analysis.

The advent and impressive progress of powerful AI models in recent years has created another important source of data from imaging. Pathologists analyze histopathological images to identify cellular abnormalities, tissue structures and patterns indicative of certain cancer subtypes. The pathologist's diagnosis will often decide the cancer therapy used, and getting it correct is a matter of life and death. The amount of data, the complexity of it and the regulatory requirements render data-management a central task of utmost importance for the success of research projects.

This thesis focuses on two main loosely connected fields of interest, NGS sequencing and AI pathology. We first describe the infrastructure that was created to support work in both those fields, with the main result being a high-availability kubernetes cluster. It gives us a common platform for containerized services and pipelines.

For NGS sequencing, we created software that applies strong cryptography to all the sequencing data we receive. This is required for human genetic data, but we apply encryption to all sequencing data by default. Data is only stored encrypted by a symmetric key algorithm, and the symmetric keys are then secured using public-key cryptography. This hybrid cryptosystem has very strong security benefits, data access requires both access to the data and holding the proper keys. Even if the servers were to be compromised the data would stay secure. Multiple potential risks were considered and mitigated. We have documented and implemented various recovery options, e.g. in case of key-loss by users. Key innovation for this software is the use of the Web Crypto API in the user's browser for all the cryptography. Users do not need to install software, and the cryptographic algorithms are built into the browser, where they are continuously audited, and browser vendors ensure their security. While our design does not require users to actively deal with the cryptography, they must also handle the data properly and secure their keys. We implement local crypto key generation, and we can save them to printed QR Codes where they closely map to the mental model for physical keys.

Connected to the NGS data, we centrally collect the relevant metadata in

a web-tool. The interface for users is similar to a simple spreadsheet, familiar to most users. But compared to simple Excel spreadsheet, through the central infrastructure, metadata is kept in sync for the whole data life-cycle. We designed metadata templates based on existing metadata standard, e.g. from the European Genome-Phenome Archive (EGA). Metadata standards are enforced in a user-friendly way by providing drop-downs and immediate user feedback on data validation. Data is stored as a full timeline of events, so even if data is accidentally overwritten it can be recovered. By having both the metadata and the sequencing data accessible through an API, pipelines can read and write the data automatically.

For AI pathology we first describe the shared infrastructure we created for our AI projects. This includes data ingestion, converting pathology slides to a common format, viewing the slides, and enabling fast random access for deep learning. We then present our work on the AI diagnosis of lymphoma on a dataset of 628 whole slides images from 157 patients. We use transfer learning to train deep neural networks on a large amount of patches, small pieces of the slide. These trained networks are then used on test data to generate diagnosis maps, locally annotated slide images that show the AI diagnosis for different regions of the slides. These maps can be evaluated by pathologists, and we can combine all the local diagnoses into a patient diagnosis. For the initial dataset of 157 patients we achieved great performance with 60% of patches classified correctly, and were able to correctly diagnose all patients. Unfortunately this performance did not transfer to an independent dataset, and more work is required to create a model that generalizes well.

Acknowledgements

This work was carried out at the Spang Lab, part of the Faculty of Informatics and Data Science (FIDS) of the University of Regensburg. I followed the PhD program of the Regensburg International School of Life Sciences (RIGeL).

First I would like to thank my supervisor, Prof. Dr. Rainer Spang for his continued support and the great opportunities and freedom to work on topics I was passionate about. I also want to thank my other supervisors Prof. Dr. Florian Hartig, for his great advice on just on the thesis but my future career and Prof. Dr. Knut Reinert for sharing his experience in the field of NGS sequencing support.

The RIGeL team and student initiatives helped me greatly in my work, and created a great way to meet and socialize with other PhD students. Especially helpful for me was Simon Holzinger and the Bioinformatics Interest Group he organizes. Through RIGeL I also met Maximilian Pichler and Lukas Heiland from Florian Hartig's theoretical ecology group, who shared the joys and struggles of coding in life sciences.

Of course, I am also grateful for the great collaboration with colleagues from my lab. I thank Dr. Jakob Simeth for an incredibly productive and pleasant collaboration on the concepts and methodology for encrypted data storage part of this thesis. Christian Kohler was a great system administrator who's work and collaboration were critical for the work in this thesis. Similarly, working with Tobias Schmidt on teaching and teaching infrastructure was a great experience with helpful discussions. Lukas Wolfseher and Dr. Paul Hüttl are great team partners for the continued research on the AI diagnosis of lymphoma. I thank Lena Buck, for great discussions in our shared office on all kinds of topics, Andreas Lösch for his great interest in science and knowledge of biology, Linda Hu for her social activism and for challenging my ideas at every opportunity, Zahra Nozari, Lukas Gessl and Christina Wüst for great discussions and input. I thank Claudio Lottaz for organizing a great Journal Club and for great input on the accessibility of my web applications. Sharon Peterson was an essential help for managing all kinds of bureaucracy, organizing great team meetings and a friendly office neighbor. Thanks also to the Oefner Group, with Prof. Dr. Oefner, Prof. Dr. Wolfram Gronwald, Dr. Katja Dettmer-Wilde, Katharina Limm, Fadi Fadil, Lisa Ellmann, Elke Perthen, Claudia Bogner and Dr. Raffaela Berger.

I also thank previous colleagues, Dr. Torsten Rehberg for great discussions on coding and free software, Dr. Gunther Glehr for great conversations and collaboration on digital infrastructure, Dr. Rudolf Schill for talks on math, physics, economics and politics and Marian Schön for teaching the Genomic Data Analysis course with me.

For research collaboration on lymphoma pathology I thank Prof. Dr. Wolfram Klapper whose skill and dedication as a pathologist enabled the project in the first place. Dr. Sarah Reinke's input was similarly important, as was her effort of scanning and digitizing all the slides. The Lab of Prof. Dr. Michael Altenbuchinger, with Stefan Schrod and Ferdinand Tolkies continue to provide great ideas and methods for solving the challenges of AI pathology. Julian Lorenz created exceptional work as a bachelors and masters student at the Spang Lab on the topic.

For the work on NGS sequencing I would like to thank fellow TRR305 researchers, Dr. Fulvia Ferrazzi and Renato Liguori for collaboration on Z01 bioinformatics. I thank Dr. Nicolas Strieder and Dr. Zbigniew Czyz for their input and expertise on metadata schemata. I thank Prof. Dr. Christoph Klein for great project coordination and for creating the valuable contact to Marcus Walther from the CDI. Dr. Melanie Werner-Klein provided great help, testing the software and giving input on the usability and regulatory requirements. For the TRR, Stefanie Pausch was also a great help, organizing meetings and retreats.

It was great to also get experience in teaching and supporting courses as part of the new FIDS faculty, working with Prof. Dr. Merle Behr, Prof. Dr. Florian Erhard, Prof. Dr. Meike Klettke and Nils Meyer.

Finally, I thank my family, Erika, Eva and Franz Huttner for their support and care for me at all times.

Contents

1 General Infrastructure

This chapter describes the general compute infrastructure that was set to support the software created as part of this thesis.

1.1 Introduction

“If you wish to make an apple pie from scratch, you must first invent the universe.”

— Carl Sagan, Cosmos

Goal of this thesis is to support research with helpful tools with a wide range of applications. While this research we do is also part of this thesis, the tools and code that support it are just as much a scientific result. We need a sort of “infrastructure for the infrastructure” that supports the creation, execution packaging and distribution of software. The following chapter describes the methods and tools used. To avoid the dilemma described by Carl Sagan we depend on many other open-source [75] projects, free and usable by everyone, instead of proprietary and closed solutions.

1.2 Containers

All this works strongly depends on *containers* for distribution, deployment and management of the infrastructure. Containers provide a standardized way to package and distribute applications along with their dependencies, ensuring portability across different environments. *Containers* in this work is used to refer to containers specified by the *Open Container Initiative* (OCI). The OCI defines two main components: the image format [51] and the runtime [52]. The image format describes how a container image should be structured. Container images are files stored in this format on the file system, and uploaded for distribution. The runtime then defines how a container should be executed, including permissions, process isolation and resource constraints. Container runtimes provide an isolated environment for apps to run, each container has its own view of the file system, its own network and its own processes. These isolation features are (typically) provided directly by the host operating system's kernel, using *cgroups* and *namespaces. cgroups* (Control Groups) limit the resource usage, e.g. CPU and memory usage. *Namespaces* give containers a separate file system and process space.

Containers can be run by themselves using a *container runtime* such as *Docker* [49] or *Podman* [77]. They will run on every operating system, but not on every *instruction set.*

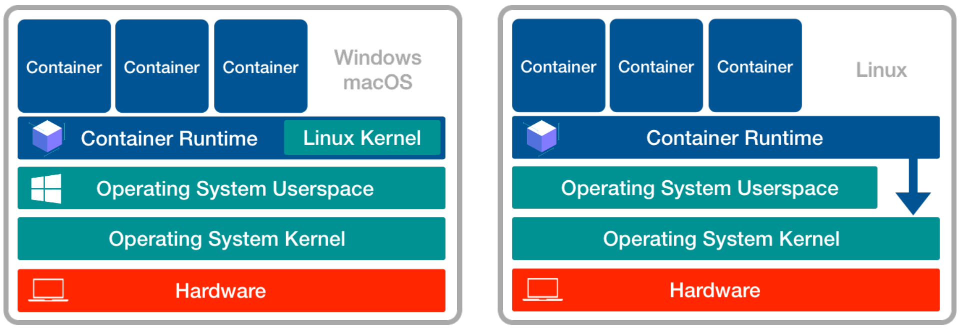


Figure 1.1: Containers on the major operating systems. OCI containers run on the Linux kernel, on Windows and macOS a virtualization layer is used to run the containers.

The more powerful application of containers is in large orchestrated cloud systems, such as *kubernetes*[2]. There containers act as the basic units of computation, they are automatically started, restarted, stopped or moved to different servers.

1.3 Kubernetes cluster

1.3.1 Hardware

Specification and setup of the Hardware was done in collaboration with Christian Kohler, the system administrator for the Spang Lab.

The work presented here does not rely on powerful compute hardware, but requires instead stability, a high fault tolerance and proper redundancy. This is why a cluster of multiple bare-metal servers was selected, running *kubernetes* [2].

Initially kubernetes was deployed to a test cluster, see 1.2 and later a highly available production cluster was set up.

Figure 1.2: Test kubernetes cluster at the Spang Lab. This cluster was used for testing before real servers in a compute-center where available. One server has a Nvidia GTX 1080Ti GPU and is used for AI workloads.

The test cluster was built from 5 old workstation PCs and one recent *“gaming”* computer with a Nvidia 1080Ti GPU. The production cluster uses three

Dell PowerEdge R740 servers as a control plane, and multiple other servers as workers. The only requirements for these servers are that they are connected via a local network connection and have a unique *MAC* address It is also important that there are at least three control-plane nodes, so a single server can be updated, or physically upgraded and there are still two other servers left that provide redundancy against hardware failure. See 1.3.6.

1.3.2 AI hardware

Key requirement for the AI workflows described in 5 were Graphical Processing Units (GPUs). AI applications, particularly training deep learning models involves performing numerous matrix multiplications and other matrix operations. GPUs can efficiently parallelize these calculations, and are orders of magnitude faster and more efficient than CPUs. Training models of even moderate size is computationally impractical without GPUs. Our models were trained initially on a Nvidia GTX 1080Ti GPU because of its cost-effectiveness. This card is not licensed by Nvidia for use in data centers and after funding was available was mostly replaced by a more powerful Nvidia A40 GPU, in one of the kubernetes nodes. With recent additional funding we also acquired a dedicated GPU server with 6 Nvidia A100 GPUs.

1.3.3 Kubernetes basics

Definitions:

- **Node:** The physical (or virtual) machine that is part of the cluster.
- **Cluster:** The set of nodes that all run kubernetes software on them. They are connected through the local network.
- **Pod:** The base unit in the kubernetes object model, a pod represents (in most cases) a single running container. Pods run on specific nodes.
- **Service:** An abstract object that describes the interface for communication with a set of pods. If we want to talk to a pod, we talk to the service and the traffic will be directed to proper node by kubernetes.
- **Deployment:** A deployment describes a *desired state* of an application. Kubernetes will try to create pods in a way to fulfill this state.
- **Ingress:** Entry and exit points to the cluster. Ingresses open network ports to the outside of the cluster, the internet (or an internal network). They then route the traffic through to the specified services.

Kubernetes is first of all a set of containers, which are distributed across multiple nodes. All the nodes communicate over the local network, they have custom cryptographic certificates, and they encrypt all inter-node traffic. On top of this, kubernetes provides a set of services, for scheduling, cluster control, networking and state management. All these services are implemented as containers running as part of the cluster.

1.3.4 Cluster state

Kubernetes tracks its own state in a shared database, called *etcd* [25]. The cluster admin communicates with the cluster through the *kube-apiserver* and specifies a *desired state*. The *kube-controller-manager* handles the core control loop of the cluster, it compares the current state against the desired state and then attempts to move the current state towards the desired state. If new pods are required the *kube-scheduler* will assign them nodes, according to pod requirements and balancing resource load.

1.3.5 Networking

In addition to running the containers, kubernetes needs to securely handle communication between containers and from the outside. Kubernetes builds its own internal network from the cluster nodes, each node, each service and each pod get its own local IP Address. Kubernetes addresses are from the special use address blocks as defined in the IPv4 Standard (RFC 791 [78]). Kubernetes runs an internal Domain Name System (DNS) for name resolution. Our cluster uses the *core-dns* service for this purpose. Each node runs *kube-proxy*, a proxy server that will redirect network traffic to the correct node. We specify a set of *edge-router* nodes, these are connected to the internet and may have open ports. These edge nodes run an *ingress controller*, which routes external traffic to the internal containers. All this networking is built on top of the same open protocols and tools that power the internet.

1.3.6 High availability

The sections 1.3.4 and 1.3.5, describe all the services required to run the cluster. They are: the *etcd* database, the *kube-apiserver*, the *kube-controller-manager*, the *core-dns*, the *kube-scheduler* and the *kube-proxy*. They are all implemented as containers and are managed by the cluster itself. All of them are crucial for cluster operation. Part of the design goal is for the cluster to be *highly available*, this means the cluster should be functional at all times.

The first step in this is creating a level of redundancy against hardware failure of the nodes. Three identical *control plane* nodes were chosen, and as long as at least one of the nodes is working the cluster will be working. A single node may be taken down intentionally for software updates or hardware fixes, with the other two still providing resiliency against hardware failure during this time.

This redundancy is achieved first by simply replicating all the core service containers on all the control plane nodes, see Figure 1.3.

But this alone does not fully solve the redundancy problems. We need to prevent a *split brain* scenario, where e.g. two schedulers disagree. In case a server fails we also need to detect the failure and redirect the traffic.

To solve both these problems an additional core service called *kube-vip* was deployed and a new virtual IP Address was created. The control plane nodes follow a leader election process where the leader takes over the virtual IP Address.

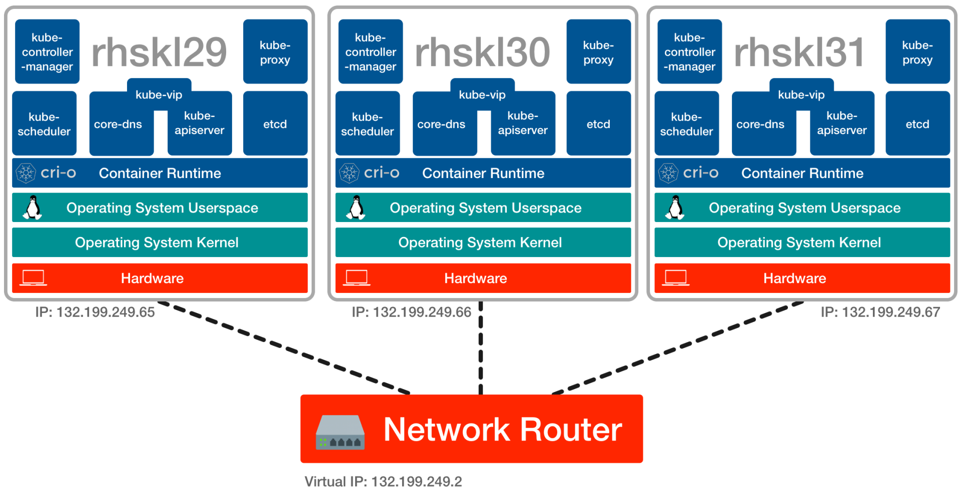


Figure 1.3: The kubernetes control plane of the Spang Lab. The control plane nodes are physical servers in the compute center. At least three control plane nodes are required to ensure high availability. All the core services are replicated on every control plane node. They use a virtual IP Address and the ARP Protocol to do leader election

1.3.7 Leader election

Leader election is done using the *Address Resolution Protocol (ARP)* [78]. ARP is the protocol that assigns physical MAC addresses of computers to IPv4 IP Addresses. The network routers will send network traffic addressed to an IP Address to the corresponding MAC address of the computer. Computers may broadcast ARP Requests in the local network to create new mappings. The leader election process for the control plane works like this:

1. We have a virtual, initially unassigned IP Address *vip*, e.g. in our case `132.199.249.2`.

2. As soon as a control-plane node starts up it waits for a short amount of time, and then sends an ARP broadcast to try to assign *vip* to their own MAC address.

3. If a node receives an ARP broadcast during the time it waited, it does not do its own broadcast.

4. The node who won the race for the first broadcast becomes the leader of the cluster, its *scheduler*, *controller-manager* and *api-server* manage the cluster state.

5. The new leader continues sending the ARP broadcast for *vip* every five seconds.

6. As long as the other control plane nodes receive the broadcast they follow the cluster leader.

7. If the leader fails, looses connection or is taken offline it can no longer broadcast ARP requests.

8. If the other nodes do not receive an ARP broadcast for five seconds they begin to broadcast their own ARP request and assume leadership of the cluster.

9. Because the IP address mapping changes, all network traffic is immediately directed to the new leader.

This process was chosen because it requires no additional hardware and can be managed from within the cluster. Failover time is at most 5 seconds.

We also deploy and ingress controller on each of the control plane nodes, and all of them are set up to act as edge routers.

1.3.8 Web security

Kubernetes may be used for any kind of network protocol, but is especially suited for applications communicating over HTTP and HTTPS (as specified in RFC 9110 [27]). All the services described later use a web based design, and they benefit from some additional services set up as part of the cluster.

Securing web traffic with HTTPS has become a requirement for modern web applications. HTTPS encrypts data transmitted between the user's browser and the web server, protecting login credentials and personal data from eavesdropping. Browsers will show warnings and disable some APIs for unencrypted HTTP connections. The Web Crypto API [33] required for the data encryption in chapter 2 is only available in secure contexts.

Fortunately *Let's Encrypt* [34], a non-profit Certificate Authority provides free certificates, through the ACME protocol [7]. As part of the cluster infrastructure the *cert-manager* tool was deployed. Cluster ingress information is already fully defined by the cluster, by associating external access points to the cluster with in-cluster services. The cert-manager service reads the cluster state, automatically generates and renews the required certificates and stores them in the cluster database. All external traffic is automatically encrypted, the ingress controller acts as an endpoint for the encryption.

1.3.9 Storage backend

Our workflows and services require a stable redundant and backed up storage. In our case large file servers, set up by Christian Kohler and Randy Rückner, serve as the storage backed. These servers manage a large amount of hard disks that are combined into a large logical volume. Data is distributed on multiple drives and parity information is created by the servers. This allows for up to two simultaneous drive failures, without data loss.

This storage volume is additionally regularly backed up on magnetic tape, this is a high-capacity but cheap way to keep data backups, and well suited for long term backups. Access is very slow but only required in case of data loss.

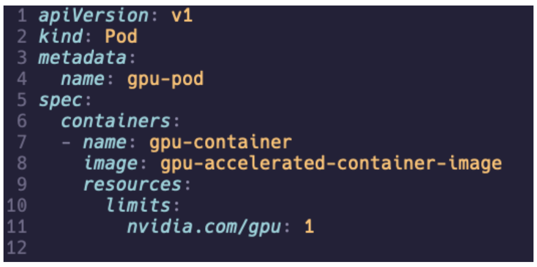
The volume is exported to all our servers using the NFS (Network File System) protocol. But the kubernetes storage model is different and built more for large cloud providers like Google Cloud, Amazon Web Services or Microsoft Azure. Kubernetes expects to create separate storage volumes, called *persistent volumes* for each service. These persistent volumes are then managed like other cluster resources. Kubernetes wants to dynamically provision storage, as in create new storage volumes on demand. This is typical for large cloud providers, storage is essentially infinite for the customer and billed by usage.

Fortunately there is software that also allows us to do dynamic provisioning with our bare-metal NFS setup. We define a custom storage *provisioner*, the *Kubernetes NFS Subdir External Provisioner* [3] maintained by the kubernetes authors. When a new persistent volume is requested, it creates a new folder inside the NFS storage as the volume and sets the required permissions.

1.3.10 GPUs as cluster resources

By default, kubernetes manages CPUs and memory as resources in the cluster. Each Pod may be assigned resource requests and limits. e.g. a Pod may request 256 MiB of *memory* and 1 *CPU* and it may have set limits of 2 GiB of *memory*. It may then use more than the requested 256 MiB, but will not be able to allocate more than 2 GiB of memory. *CPU* as kubernetes resource refers to the number of CPU cores, and a request of 0.5 CPUs is a request for half the CPU time of a single core. The kubernetes scheduler uses these limits and requests to distribute services according to their resource needs across different nodes.

For our GPUs the Nvidia Container Toolkit [73] allows us to generate Container Device Interface (CDI) specifications [17]. This allows the scheduler to manage not just CPUs and memory, but also GPUs as a cluster resource. Pods and Jobs can simply be annotated, see Figure 1.4, and will be scheduled correctly. With just the resource annotations, kubernetes also knows that the GPU device should be mounted into the container.



```yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod
spec:
  containers:
  - name: gpu-container
    image: gpu-accelerated-container-image
    resources:
      limits:
        nvidia.com/gpu: 1

```

Figure 1.4: A kubernetes Pod specification with a request for a GPU resource, it will automatically be scheduled to a node which contains a GPU.

1.3.11 Kubernetes jobs

The kubernetes cluster mostly manages services, these are expected to stay running, and they are restarted automatically should they crash. But data pipelines are not services, they should be run once, and they should terminate. For this use case kubernetes provides a different API, the *kubernetes Job API*. A Job represents a finite task that runs to completion, the Job API is designed to also run Pods, but then checks if they successfully terminate. Jobs may specify parallelism, how many jobs should run at the same time in the cluster, and a back-off limit, how often the Job should be retried in case of failure.

With this API we created containerized data pipelines, and schedule them to run on the cluster. AI pipelines may simply request a GPU, see 1.3.10, and will then run on a GPU node. These pipelines are reproducible, the containers are built automatically from code repositories using our CI, see 1.6.4 and stored in our container registry.

1.4 Authentication and authorization

For user authentication a custom authentication provider was set up. It acts as both a provider for the *OpenID Connect Protocol* [30] and as an *LDAP* [106] server. This service manages all the personal user data, and complies to the General Data Protection Regulation(GDPR) [26]. The minimally required data is collected, and a privacy policy is provided to users. Using the OpenID Protocol, other services only request the required scopes and users get a clear prompt each time personal data is requested.

1.4.1 Access control

For authorization, we define set of permissions as part of the authentication provider. For each service users may be granted permission to use the service, and in addition may be granted additional *admin* permissions.

1.4.2 OpenID Connect

User information and permissions are exported first using the OpenID Connect protocol [30]. This protocol evolved from *OAuth2* and is by far the most popular authentication method on the internet.

It is used by applications to securely outsource authentication to identity providers, and well known through sign-in flows like *Sign-in with Google*, *Sign-in with Apple*, *Sign-in using Facebook* etc.

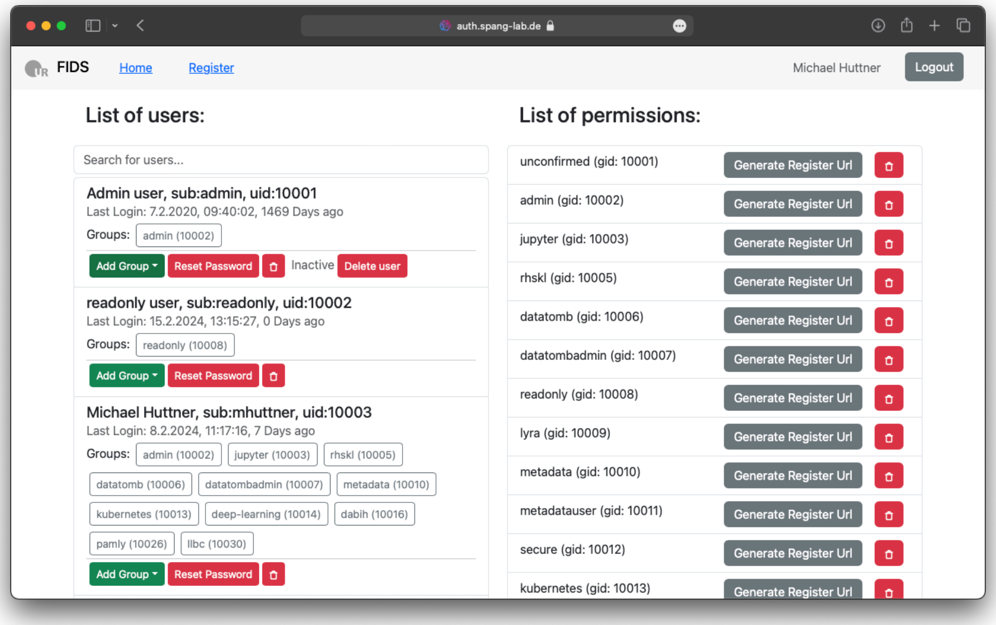
The OpenID Connect flow works like this

1. The user navigates to a website or web application, called the Relaying Party (RP)

2. The users clicks *Sign-In*

3. The user is redirected to the Identity Provider (IP).

4. The RP requests a certain set of scopes from the IP, these scopes represent certain user information, such as the name, email, address or permissions.



List of users:

Admin user, sub:admin, uid:10001
Last Login: 7.2.2020, 09:40:02, 1469 Days ago
Groups: admin (10002)

readonly user, sub:readonly, uid:10002
Last Login: 15.2.2024, 13:15:27, 0 Days ago
Groups: readonly (10008)

Michael Huttner, sub:mhuttner, uid:10003
Last Login: 8.2.2024, 11:17:16, 7 Days ago
Groups: admin (10002) jupyter (10003) rhskl (10005) datatomb (10006) datatombadmin (10007) metadata (10010) kubernetes (10013) deep-learning (10014) dabih (10016) pamly (10026) llbc (10030)

List of permissions:

unconfirmed (gid: 10001)
admin (gid: 10002)
jupyter (gid: 10003)
rhskl (gid: 10005)
datatomb (gid: 10006)
datatombadmin (gid: 10007)
readonly (gid: 10008)
lyra (gid: 10009)
metadata (gid: 10010)
metadatauser (gid: 10011)
secure (gid: 10012)
kubernetes (gid: 10013)

Figure 1.5: Auth server user management. Authorization is managed by a set of permissions per users. These permissions are also mapped to LDAP groups, and can be used to set file system permissions.

5. RPs should request the minimal required set of scopes.

6. The IP authenticates the user, e.g. with the user entering his username as password.

7. The user must consent to grant the RP access to the requested scopes.

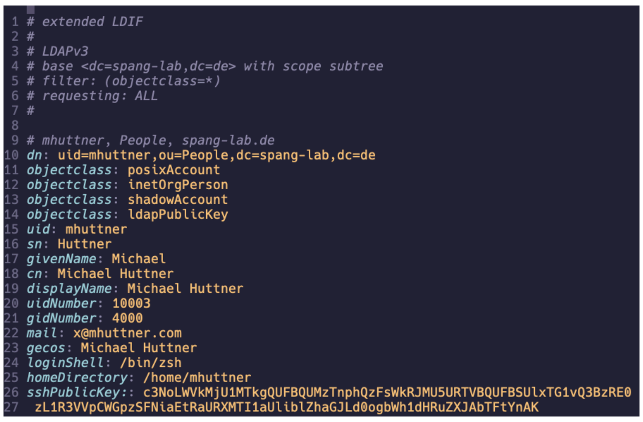
8. The IP sends an Identity Token and an Access Token to RP.

9. The RP sends a request to the userinfo endpoint of the IP to fetch the request user data.

For our applications we depend on this flow, and we use our custom identity provider to store the user data. Because we externalize authentication, our software is easy to reuse by others, they simply configure their own Identity Provider, or use a public Provider like Google, Apple, Facebook, Otka, or GitHub.

1.4.3 LDAP mapping

Because we also support Linux compute environments, see 1.5.1, we use the user data in our identity provider to emulate an LDAP server. LDAP, the Lightweight Directory Access Protocol, organizes user data in a hierarchical tree structure, and maps well to the Linux user model of *users* and *groups*. We assign each permission, set in 1.4.1 an LDAP group and ID, this can then be mapped to file system permissions in our compute environment. Note that the there is only a single source of truth for the user data, but it is exported for both the OpenID and the LDAP protocol. These two protocol cover all the authentication needs for our infrastructure. The SAML [13] protocol, was not implemented but may be useful in the future. LDAP access is limited to within the cluster and a valid bind is required to read user data. The LDAP tree is read only, write operations will fail. Changes to the user data are done using the HTTPS API or our identity provider, mostly from the web client, see Figure 1.5.



```
# extended LDIF
#
# LDAPv3
# base <dc=spang-lab,dc=de> with scope subtree
# filter: (objectclass=*)
# requesting: ALL
#

# mhuttner, People, spang-lab.de
dn: uid=mhuttner,ou=People,dc=spang-lab,dc=de
objectclass: posixAccount
objectclass: inetOrgPerson
objectclass: shadowAccount
objectclass: ldapPublicKey
uid: mhuttner
sn: Huttner
givenName: Michael
cn: Michael Huttner
displayName: Michael Huttner
uidNumber: 10003
gidNumber: 4000
mail: x@mhuttner.com
gecos: Michael Huttner
loginShell: /bin/zsh
homeDirectory: /home/mhuttner
sshPublicKey:: c3NoLWVkMjU1MTkgQUFBQUMzTnphQzFsWkRJMU5URTVBQUFBSUlxTG1vQ3BzRE0
 zL1R3VVpCWGpzSFNiaEtRaURXMTI1aUliblZhaGJLd0ogbWh1dHRuZXJAbTFtYnAK
```

Figure 1.6: LDAP export of user data.

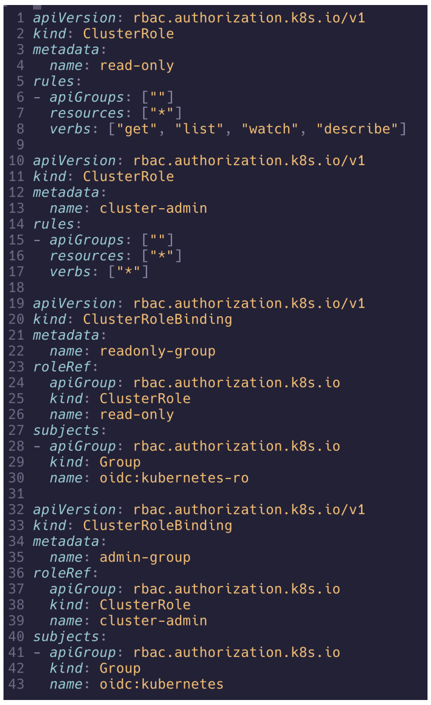
1.4.4 Cluster authentication

Even thou our identity provider runs as a service in the cluster, we can also use it to manage permissions for cluster administration. Kubernetes supports the OpenID Connect token authentication [4]. We configure the kubernetes API servers to accept access tokens from our identity provider and then users can use *kubelogin* [54] to sign in and set the access token for the kubernetes command line tool *kubectl*. Kubernetes uses role base access control (RBAC) to manage cluster permission. Cluster *roles* define permissions for actions in the cluster. By defining *ClusterRoleBindings* we can map users and groups to *roles*. This allows for fine-grained access control for cluster management, see Figure 1.7.

1.5 Kubernetes services

1.5.1 Compute servers

Even with a reproducible job queue, described in 1.3.11, some use cases are better served by dedicated server. Especially for testing and exploratory analysis, the overhead of creating a container creates too much friction and prevents rapid iterations of code. Debugging also requires more active control



```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: read-only
rules:
- apiGroups: [""]
  resources: ["*"]
  verbs: ["get", "list", "watch", "describe"]

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-admin
rules:
- apiGroups: [""]
  resources: ["*"]
  verbs: ["*"]

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: readonly-group
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: read-only
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: oidc:kubernetes-ro

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-group
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: oidc:kubernetes
```

Figure 1.7: Kubernetes configuration for defining cluster roles and permissions. This maps the group *kubernetes* to the cluster admin role with full permissions, and also defines a read-only role, that may only read, but not change cluster state.

and monitoring of the pipeline. Dedicated machines would typically be virtual or physical machines, with ssh access for users running a *headless* Linux operating system. Users work on these servers the same way as their local machine, they can install software, transfer files and run code. They connect using the secure shell (ssh) protocol, a cryptographic network protocol, and all the communication is encrypted. This way of remote work is wide-spread and familiar to users. We wanted this, more direct, way of interacting with the machines without compromising the security or stability of the cluster. To solve this we created custom containers, based on Ubuntu [15] a full server operating system. These containers can be scheduled on the kubernetes cluster like any other service and they can request GPUs. We run the OpenSSH [97] server *sshd* as the main container process and enable ssh access to the cluster. Only public-key based authentication is permitted, and the public keys are managed by the identity provider, see 1.4. We can offer different containers based on workload and still have an effective sandbox where applications can run without affecting the cluster. Because these containers access the host systems kernel we do not lose compute performance from virtualization, as it would be the case in dedicated VMs. These compute containers are in active use not just for this work, but they support a variety of scientific work from the Spang Lab. Each user has an individual persistent storage claim mounted via NFS as described in 1.3.9.

1.5.2 Jupyter

Jupyter Notebooks are widely used for various tasks such as data exploration, prototyping, model training and result visualization. Jupyter is an open-source interactive web application that enables users to create and share documents, called *Jupyter Notebooks* that contain code, equations, visualizations and text. Jupyter was set up on kubernetes, thanks to the open-source project *Zero to JupyterHub with Kubernetes* [80]. This was initially created at the UC Berkeley for use in their *Data8* [102] teaching program, but is now officially supported by Project Jupyter. This project integrates easily with our infrastructure, with the notebooks running as Pods on the cluster, user authentication bound to our identity provider, see 1.4 and storage dynamically allocated from our file server, see 1.3.9.

GPUs can be mounted as resources into the notebook pods, see 1.3.10 creating an easy way to train our neural networks, see 5 and evaluate results. This was especially useful for students working on the AI project, who could quickly start working on AI workflows.

Jupyter Notebooks were not only useful for the work in this thesis, they were also an essential tool for teaching at the Spang Lab. We could offer the course *Genomic data analysis* [92] with a custom coding environment, based on Jupyter and the *Data8* [102] program, for students unfamiliar to coding and statistics. Containers (1.2) provided a working coding and teaching environment for students. Jupyter enabled us to teach an interactive coding course remotely during the COVID-19 pandemic. As part of the Faculty for informatics and data science (FIDS) Jupyter is an official part of the *Data Science* study program, with a course *Developer Skills* [95] created by me and the other PhD Students of the faculty. Infrastructure and teaching support for this and the *Data* course was provided by Tobias Schmidt and me, in collaboration with the FIDS administrator Nils Meyer and the computing center of the University of Regensburg. Tooling for this, called *urnc* is already published on GitHub [45], but not directly related to the work described in this thesis.

1.5.3 Other scientific web services

This cluster also serves as basic infrastructure for other projects.

- **Lyra** [46]: Data exploration and visualization of lymphoma expression and suvival data.
- **DTD** [89]: Web app for digital tissue deconvolution.
- **Celloscope** [88]: Visualization of scRNA seq data.
- **Datatomb** [55]: Scientific data management.

1.6 Code and container management

1.6.1 Git repositories

The work in this thesis is in large parts, code written in different programming languages and for different projects. For years the state of the art in code management is *Git* [36]. Because code is plain text and requires little storage we can keep a complete history, every version with every change, for all code we write. Git is a distributed version control system design to track these changes in source code. It was created by Linus Torvalds in 2005 for use on the Linux kernel. All projects in this thesis are one or more git repositories (repos). Repositories are collections of files with their revision history. This history is organized as a tree of *commits*, snapshots of the project at a specific points in time. All code is hosted and backed up, either on our own self-hosted instance of *GitLab* [11] or is publicly available on *GitHub* [50].

1.6.2 Semantic versioning

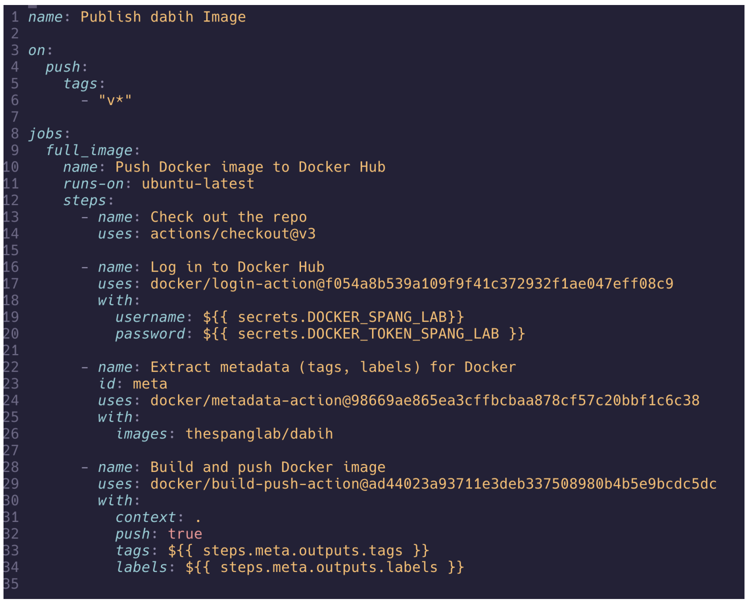
Stability and reproducibility is important for the software created and published as part of this thesis. For all of our software projects and containers we use a versioning scheme that describes the nature of changes and whether they are compatible with existing code or introduce breaking changes. This is why we use the Semantic versioning (SemVer) versioning scheme [79]. Each code and container release has a version number with 3 numbers in the format `MAJOR.MINOR.PATCH`, e.g 1.0.4. After changes this version is updated, depending on the extent of the changes. Incrementing the `PATCH` version, e.g. 1.0.4 $\Rightarrow$ 1.0.5 indicates bug fixes or small improvements that are fully backwards compatible. Changing the `MINOR` version implies some breaking changes, and the `MAJOR` version is increased when there are fundamental changes to the software. The version tags are attached to commits in the version control system, see 1.6.1, and the software code can be reverted to a certain version with a simple git command e.g. `git checkout v1.0.4`. Version tagged containers are stored in the container registry, see 1.6.3, and create a fully reproducible environment. This versioning is essential for reproducible and reusable science, and are part of the FAIR guiding principles [104].

1.6.3 Container registry

Similarly to the code management described in 1.6.1, we also store the containers we create. Because of the features described in 1.2, containers are a key technology for reproducible research [8]. We not only keep the code, see 1.6.1, but also the whole environment, with all dependencies. The Spang Lab hosts an internal container registry, a service that stores and distributes containers as part of the GitLab [11] instance, maintained by Christian Kohler. In addition, for open-source web services we also publish containers on the public container registry *DockerHub* [48].

1.6.4 Continuous integration

We have automated the container and software build and publication process using the continuous integration (CI) and continuous deployment (CD) platforms of our GitLab server and of GitHub. These automations define the containers only from specific commits. Git commits tagged with a semantic version, see 1.6.2, trigger the pipelines. For an example see Figure 1.8. For our GitLab server we set up a custom runner on a Linux PC to execute the pipelines and build the containers. GitHub Actions provides free runners with macOS (ARM and x86), Linux and Windows machines in the cloud for open-source projects. With this we can create build matrices and easily create cross-platform binary releases for our software, see 1.9.



```yaml
name: Publish dabih Image

on:
  push:
    tags:
      - "v*"

jobs:
  full_image:
    name: Push Docker image to Docker Hub
    runs-on: ubuntu-latest
    steps:
      - name: Check out the repo
        uses: actions/checkout@v3

      - name: Log in to Docker Hub
        uses: docker/login-action@f054a8b539a109f9f41c372932f1ae047eff08c9
        with:
          username: ${{ secrets.DOCKER_SPANG_LAB}}
          password: ${{ secrets.DOCKER_TOKEN_SPANG_LAB }}

      - name: Extract metadata (tags, labels) for Docker
        id: meta
        uses: docker/metadata-action@98669ae865ea3cffbcbaa878cf57c20bbf1c6c38
        with:
          images: thespanglab/dabih

      - name: Build and push Docker image
        uses: docker/build-push-action@ad44023a93711e3deb337508980b4b5e9bcdc5dc
        with:
          context: .
          push: true
          tags: ${{ steps.meta.outputs.tags }}
          labels: ${{ steps.meta.outputs.labels }}
```

Figure 1.8: GitHub action that automatically builds and publishes the dabih container from a commit that is tagged with a semantic version.

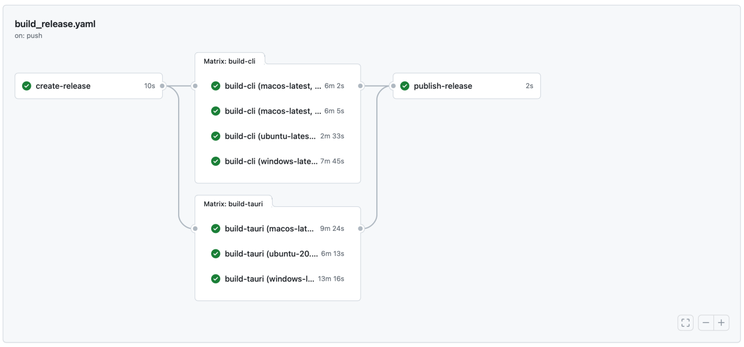

build_release.yaml
on: push
create-release 10s
Matrix: build-cli
build-cli (macos-latest, ... 6m 2s
build-cli (macos-latest, ... 6m 5s
build-cli (ubuntu-lates... 2m 33s
build-cli (windows-late... 7m 45s
Matrix: build-tauri
build-tauri (macos-lat... 9m 24s
build-tauri (ubuntu-20.... 6m 13s
build-tauri (windows-l... 13m 16s
publish-release 2s

Figure 1.9: The GitHub build matrix for releasing precompiled versions of dabih (see chapter 2). GitHub provides free runners for MacOS (x86 and ARM), Linux and Windows systems.

1.7 Web development

Web services have long been essential for NGS workflows, with web applications such as the Ensembl genome browser [66], the databases from the NCBI [86] and many others. These tools are generally built for bioinformaticians, while they may offer some services in the browser they act more like a data source that is consumed using a programming language, such as R or Python. The services we built also enable this kind of workflow, through the use of machine-readable JSON web APIs, see 1.7.2, but the evolution of web technologies in the recent years enables us to provide powerful new applications in the browser.

1.7.1 Databases

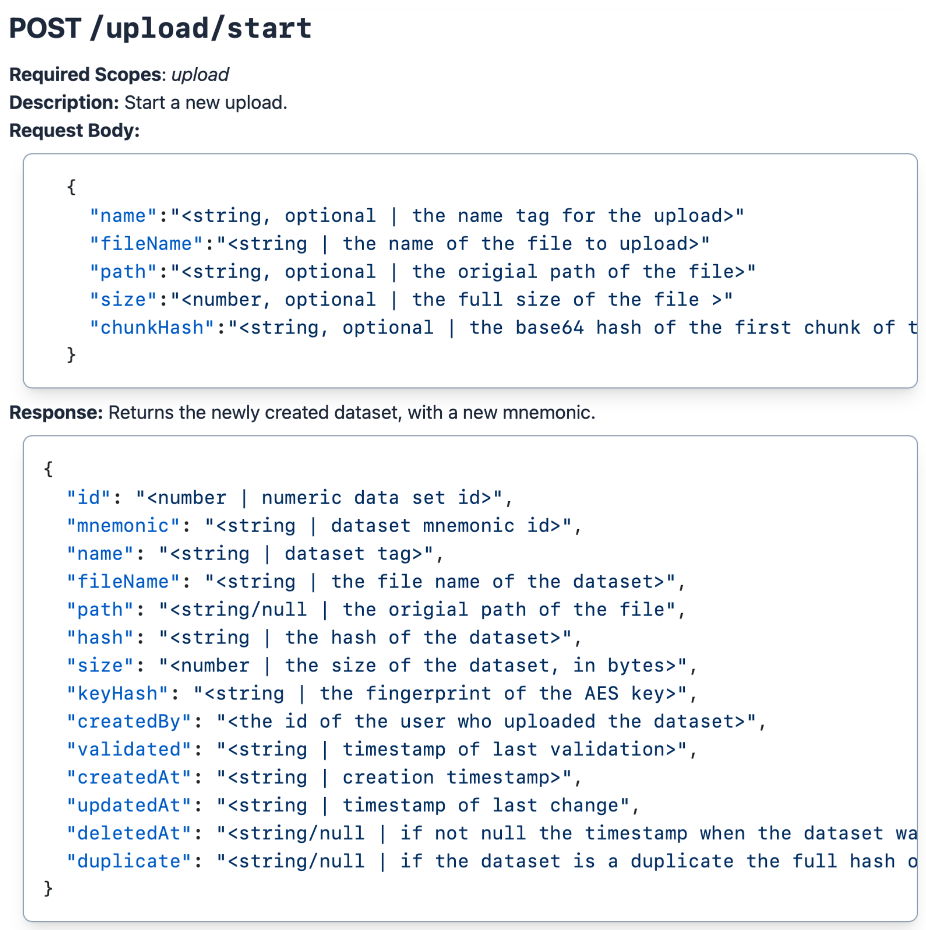
We follow best practice for storing all of our data, while large datasets are stored directly on the file system, all the metadata is stored in databases based on the Structured Query Language (SQL). We set up a highly available instance of PostgreSQL [99] on the cluster hardware. This works similarly to the high availability of the cluster, see 1.3.6. There is a primary database server, that manages all the writes to the database and acts as a source of truth for read-only replicas. Reading data from the database can happen from either the primary or a replica. We use a smart Object Relational Model (ORM) that maps well to many SQL based database technologies, called *Sequelize* [18]. This allows us to share our software without requiring a specific database technology as a dependency. By also supporting SQLite [93] we only require some kind of persistent storage, with no separate database server.

1.7.2 JSON API

The make the data in these databases accessible, with proper access control, and do the processing required we use API web servers based on NodeJS [98]. These web servers do not have a graphical user interface, or render HTML. They use JSON [9] as their data format for communication, see Figure 1.10. JSON is a Lightweight data interchange format that is easy for both humans to read and write, and easy for machines to generate and parse. JSON APIs have largely replaced XML and are the primary way to exchange data over the internet. API requests over HTTP are stateless, meaning self-contained and independent, and will be handled asynchronously by the web server. This means instead of waiting for a single request to finish the server can process multiple requests concurrently. This is different from parallelism, because concurrency happens on a single thread but instead of blocking execution, e.g. when waiting for a database request or file system read, the server will continue with other tasks, and be notified upon completion.

1.7.3 Web clients

In recent years the browser has evolved from an application that simply renders HTML received from the server to a full cross-platform application platform. Code running in the browsers is no longer simple DOM manipulation, but



POST /upload/start

Required Scopes: *upload*
Description: Start a new upload.
Request Body:

```
{
  "name":"<string, optional | the name tag for the upload>"
  "fileName":"<string | the name of the file to upload>"
  "path":"<string, optional | the origial path of the file>"
  "size":"<number, optional | the full size of the file >"
  "chunkHash":"<string, optional | the base64 hash of the first chunk of t
}
```

Response: Returns the newly created dataset, with a new mnemonic.

```
{
  "id": "<number | numeric data set id>",
  "mnemonic": "<string | dataset mnemonic id>",
  "name": "<string | dataset tag>",
  "fileName": "<string | the file name of the dataset>",
  "path": "<string/null | the origial path of the file",
  "hash": "<string | the hash of the dataset>",
  "size": "<number | the size of the dataset, in bytes>",
  "keyHash": "<string | the fingerprint of the AES key>",
  "createdBy": "<the id of the user who uploaded the dataset>",
  "validated": "<string | timestamp of last validation>",
  "createdAt": "<string | creation timestamp>",
  "updatedAt": "<string | timestamp of last change",
  "deletedAt": "<string/null | if not null the timestamp when the dataset wa
  "duplicate": "<string/null | if the dataset is a duplicate the full hash o
}
```

Figure 1.10: A documented JSON API call for the dabih (see chapter 2) application. This documents the http endpoint for starting a new upload `/api/v1/upload/start`. The client sends a JSON Request and receives a JSON response from the API server.

it has access to many standardized browser APIs. These functionalities make browsers a viable platform for more and more use cases, with major advantages for software distribution, portability and ease of use. In chapter 2 we describe a novel use of the Web Crypto API [33], which only became possible after adoption from browser vendors.

With this new complexity, new tooling and methodology is required to build these web applications. While language of the web, JavaScript, is an interpreted language without a compiler, transpilers play a crucial role. Transpilers compile source code to source code, and their primary purpose is to enhance compatibility. They also perform tasks such as code optimization, and they introduce new language features. Modern web applications now have a full build and development system.

Software development also changed to different architectures based on *Components*, with *React.js* [68] as the most popular library. React components allow us to create modular, reusable and composable pieces of the user interface.

For our web clients we use the *Next.js* [103] framework that integrates React components and uses Webpack [58] as a build tool.

Usability is a key concern for our applications, and having consistent styling and the proper signifiers for interactions is important. With the proliferation of mobile devices we also needed to adopt responsive design principles, with the graphical interface automatically adapting based on screen size and layout. We rely on Tailwind CSS [62] for styling our web applications, a set of low level utility classes to build designs directly in the web component markup.

2 dabih - encrypted data storage and sharing platform

Parts of this chapter are published as preprint [47]. Other authors and their contributions:

Michael Huttner: Conceptualization, Methodology, Software, Writing - Original Draft, Writing- Review & Editing, Visualization; **Jakob Simeth:** Conceptualization, Methodology, Software, Writing - Review & Editing; **Renato Liguori:** Validation, Writing - Review & Editing; **Fulvia Ferrazzi:** Validation, Writing - Review & Editing, Supervision; **Rainer Spang:** Writing - Review & Editing, Supervision; The text and visualizations in this thesis are my original work.

2.1 Introduction

Modern biomedical research relies heavily on large datasets, acquired by various techniques such as sequencing analysis or imaging. This encompasses the acquisition, storage, sharing and analysis of highly sensitive data, including human genomic data. Handling such data carries significant ethical and legal implications, which are governed by regulations like the General Data Protection Regulation (GDPR) in the European Union. Researchers must maintain stringent security measures and uphold confidentiality to protect the integrity of sensitive data. For most sensitive clinical data, proper anonymization or pseudonymization are effective and practical solutions to protect the individual's privacy. But genomic data is special because it is identifiable by nature. In this case, the principle of least privilege [84] must be rigidly applied. This can be achieved through the use of asymmetric encryption, limiting access to a minimal set of authorized individuals. Additionally, implementing fine-grained access control further ensures that only those authorized individuals can access the data. Software and algorithms for this purpose are well established, with comprehensive recommendations available, such as those from the German Federal Office for Information Security [29]. The predominant shortcoming is the usability of these algorithms especially in integrating key management, authentication, and authorization. For example, the most widely used standard OpenPGP [28], implemented by the GnuPG software, requires installing software, cryptography knowledge and is built for use in the command line. Typically, data owners are clinicians and biomedical researchers who may not possess extensive IT expertise. It is crucial for them to manage their data securely while avoiding the complexities involved in understanding encryption and key management in detail. To address this, here we present dabih, an open-source web application specifically designed to facilitate user-friendly encrypted data management. dabih relies on the *Web Cryptography API* [33], a tool integrated in modern web browsers that allows us to overcome many usability and portability issues, by employing a web application that functions within the browser.

2.2 Methods

2.2.1 Preliminary work

Dabih is based on a simpler unencrypted data storage, conceptualized and implemented in collaboration with Jakob Simeth. This work is also available open-source on GitHub, as

- **datatomb** [55]: The data storage API server.
- **glacier** [56]: A command line interface for datatomb.
- **diggeR** [91]: A simple R package for interfacing (searching, downloading, uploading) with datatomb.

While datatomb implemented data storage, it did not have a web client. It was intended mostly for use in the command line, and in data analysis pipelines written in the *R* language [100]. It also did not support encryption, but included some metadata tracking that was excluded from dabih.

2.2.2 Design

dabih implements a hybrid cryptosystem with symmetric-key encryption for data and public-key encryption as key encapsulation mechanism, enabling easy permission changes by re-encrypting only the symmetric key to authorized data recipients. The 256-bit Advanced Encryption Standard with Cipher Block Chaining (AES-256-CBC), as specified in NIST SP800-38A[70] is used as the symmetric algorithm, 4096-bit RSA (Rivest – Shamir – Adleman) with Optimal Asymmetric Encryption Padding (OAEP) as specified in RFC3447[83] is used for key encryption.

dabih is implemented as a server-client system, see 1.7. The server (we call *dabih server*) provides a web API, receives and sends the data and manages the keys. Clients handle the cryptography related to the private key on the user's device, e.g. the user's web browser.

2.2.3 Authentication and key management

For authentication, dabih uses third-party services through the *OAuth2* protocol. This allows server administrators to configure various authentication providers such as Google, GitHub, Keycloak, or any other OpenID provider, facilitating easy integration with different institutional setups, see 1.4. dabih requests a user ID, name, and email from these providers. By default, anyone can sign up; however, to actively use dabih, users must also submit an RSA

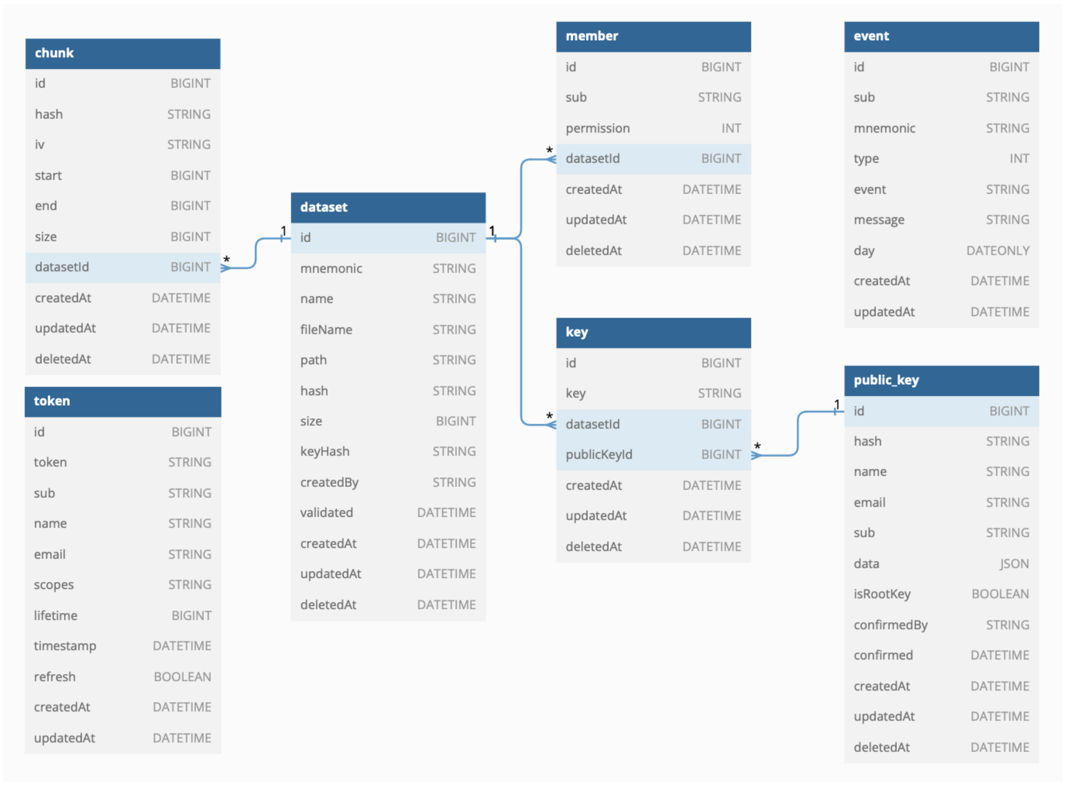


Figure 2.1: Database structure for the dabih.

public key. This key requires admin confirmation before it becomes active, ensuring that admins maintain control over who can access their dabih instance. The RSA key-pair can be generated by the dabih client or by using the popular tool *ssh-keygen*

```
ssh-keygen -m pkcs8 -t rsa -b 4096 -f key.pem
```

dabih requires a 4096-bit RSA key-pair in the PKCS#8 format. In the following we assume users have a valid RSA key-pair and have uploaded their public key to dabih. We stress that the private key never leaves the data owner's hardware and is not uploaded to the server.

2.2.4 Uploading data to dabih

In the context of dabih, 'datasets' always refer to single files. For directories, clients will convert them into archive files prior to uploading. The upload process in dabih begins with the client initiating the action by first sending file metadata to the upload API endpoint. The server will then generate an ID and a cryptographically strong pseudo-random 32 byte AES-256-CBC key. The ID will be returned to the client, the key will be kept in memory until the upload is complete, it is not written to disk. The AES Key is encrypted with the RSA public key of the user uploading the data and then stored in the database. The dabih client will then start the data upload by splitting the file into chunks (typically 2 MiB) and sending them to the server. For each chunk the server will generate a random 16 byte initialization vector (IV)

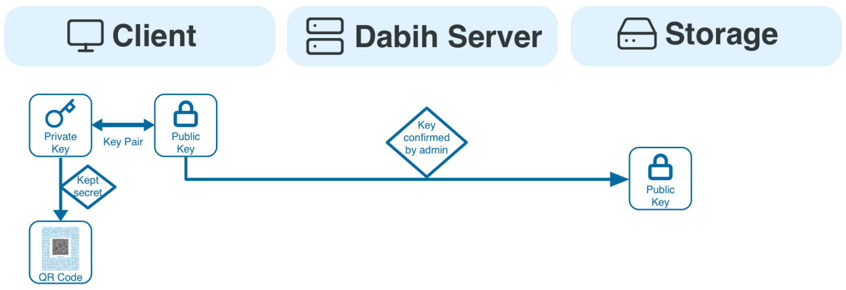


Figure 2.2: RSA-4096 key-pair generation and upload to dabih. Key generation always happens locally, and the private key never leaves the user's computer. The public key is uploaded to dabih. The private key can be generated as a printable QR Code and scanned in with a normal webcam, it will only be stored in the browsers local storage.

and encrypt the chunk with the IV and the AES key. Only after encryption, the data is written to the storage backend, fully implementing data-at-rest encryption. dabih also generates an SHA-256 hash of the unencrypted data, which allows us to check and skip duplicate files as well as resume incomplete uploads. A checksum of the encrypted data is calculated using the CRC-32 algorithm [61], this is an emergency redundancy against data corruption, but it is important that the storage used has adequate protection and redundancy by itself.

After all chunks have been uploaded dabih calculates a dataset hash by concatenating the bytes of all the chunk hashes and hashing this data again with SHA-256. We also write a recovery file to the storage, which can be used for offline data recovery, see 2.2.10. See Figure 2.3 for a schematic overview on this process.

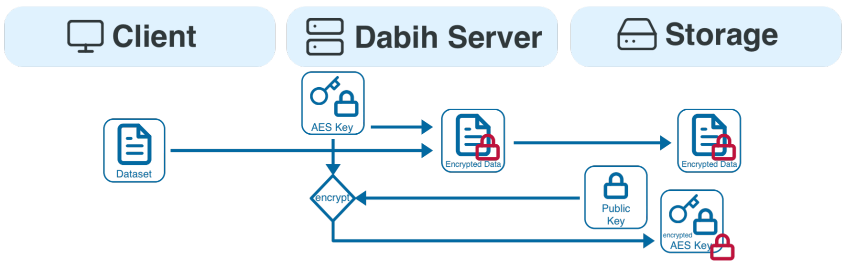


Figure 2.3: The dabih upload process: When data is uploaded to dabih we generate a random key in memory and use symmetric encryption (AES-256-CBC) to encrypt data before it is stored. Dabih loads the public key of the uploader from its database. With this public key we can complete the upload and encrypt the AES key. The encrypted key is stored and can only be decrypted using the private key that dabih does not have.

2.2.5 Data sharing

Data owners can grant authorized recipients either *read* or *write* access to their datasets. *Read* access permits downloading the data, whereas *write* access provides more extensive permissions. With *write* access, recipients can further share access with additional users, re-encrypt the dataset, and even delete it.

Assume *User A* wants to share a dataset with *User B*. To do so, *A* downloads the encrypted AES key for the dataset. Locally *A* decrypts the key thus obtaining an unencrypted AES key for the dataset. User A uploads this key again, the server does not hold a copy of it, and the server re-encrypts it with the public key of *B*. To prevent key exchange attacks from *A*, the server compares the SHA-256 fingerprint of the AES key to its database and rejects the key if it does not match. This happens for both the *read* and *write* permission, the different permission levels are written to the database and checked on API calls by the server. This process is visualized in Figure 2.4.

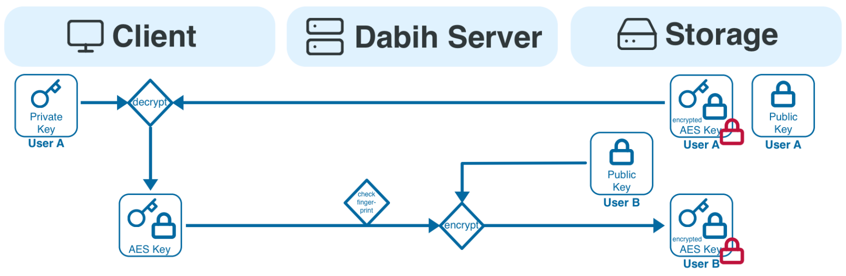


Figure 2.4: The dabih data sharing process: Because dabih itself cannot access the data, only a user who already has access can share the dataset with others. Data sharing is similar to downloading, but only the AES key is downloaded. This key is then sent back to dabih and encrypted with the public key of the new user.

2.2.6 Data download

Similarly to sharing data access, downloading datasets starts by downloading and decrypting the encrypted AES key. Now the client can simply download the encrypted data and decrypt it locally, as seen in Figure 2.5. While this is the most secure way to download data, users can also send the decrypted key to dabih. The server will then decrypt the dataset and send the raw data, offloading this computation from the client system. Data is still encrypted in transit by the https transport layer security (TLS) and clear-text chunks are kept only in memory.

2.2.7 Data ingestion

A ’side effect’ of this encryption scheme is that, technically, no private key is required for uploading data to dabih. This creates a way of data ingestion: users can allow others to upload data into their account. Dabih enables this

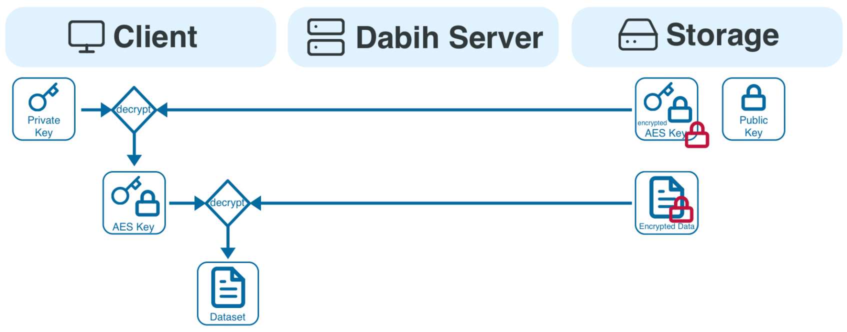


Figure 2.5: The dabih download process: a two-step decryption process is required. First the encrypted AES key is downloaded and is decrypted using the private key. This results in the unencrypted AES key. Then the encrypted dataset is downloaded and decrypted using the newly acquired AES key.

through the use of upload tokens. Each user can create one of more access tokens on their account page. This token can then be sent to others with a special link. This link shows a modified upload page that uses the token owners public key for encrypting the data. No cryptographic key and no account is required to upload. This can be very useful for securely collecting data from others or from automatic processing pipelines, e.g. a link can be sent to a sequencing provider to upload data directly, no other software or account required. Unfortunately web browsers have limitations for ingesting large amounts of data, e.g. it has to stay open during the whole upload and our code can only access files directly selected by the user. This is why we created an optional application for data ingestion as part of dabih 2.3.3, it can be downloaded and run to upload large amounts of data to dabih. These upload tokens are not very secure as they are part of the URL, which is why they are scoped only to upload API calls, and will expire after some time if not renewed by the user.

2.2.8 Key loss

Datasets remain encrypted throughout their entire tenure on the server, ensuring that only authorized data owners have the capability to decrypt them. This approach eliminates the possibility of a central authority, like a system administrator, recovering data if a user's private key is lost or stolen. Nevertheless, if at least one other user maintains access to the data, it can be re-encrypted, which allows for the restoration of access. Re-encryption is done by first decrypting the AES key for the dataset by the user who still has access to the data, or by using a root key, see 2.2.9. This key is sent to dabih, which generates a new random AES key. All the data chunks are decrypted and then re-encrypted with the new key. This new key is then encrypted to all the public keys of users who are authorized to have access to the dataset. See Figure 2.6. There is no disruption to other users, as long as they did not store or cache the old keys or data locally.

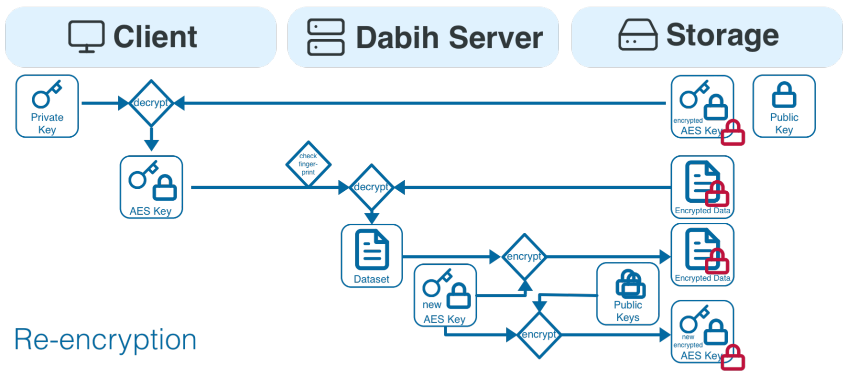


Figure 2.6: Dataset re-encryption in case of key loss. As long as some other user has access to the data it can be re-encrypted. The user with access download the encrypted AES key, decrypts it and sends it to the server. The server uses the key to decrypt the dataset, generates a new AES key and then re-encrypts the dataset with the new key. All existing access permissions stay intact, the public keys are known to dabih and are used to generate new encrypted AES keys.

2.2.9 Root keys

If all users with access to a dataset lose their private keys, the data becomes irrecoverable. This scenario is particularly likely if certain datasets are accessible to only a single user. To address this issue, dabih incorporates *root keys* as an emergency backup solution, providing a safeguard against such situations. Root keys are ordinary RSA-4096 key-pairs, just as every user key. One or more public root keys can be configured for the dabih server, and new datasets will be automatically encrypted to every public root key. Since root keys bypass the security system, private keys must be stored in a physically secure location with strict access controls and only used for emergency recovery of the data in case all other keys are lost. The code for recovering datasets this way is part of the dabih command line interface, see 2.3.2.

2.2.10 Offline recovery

Another disaster scenario is the loss of the dabih database. We stress that storage must be backed up independently. As a precaution against loss of the database we write the most important recovery data to disk as a part of the dataset. This recovery file contains a list of all chunks in a dataset, with their hash, crc32 checksum and AES initialization vector. Also included is the AES key encrypted with each public root key. This is all the information required to decrypt the data with one of the private root keys.

2.2.11 Mnemonic based ID system

Dabih needs a way to uniquely identify datasets, and these identifiers will be exposed to users. As part of the design we decided on human friendly identifiers for dabih, we call them *mnemonics*. A dabih mnemonic identifier will typically be a random adjective combined with a random first name. e.g. `vampiric_aviyana`, `unsaluted_esmerelda` or `branchless_eliyana`. dabih ensures uniqueness, and its name database currently contains 28476 adjectives and 101337 first names, allowing for up to 2.8 billion datasets. Mnemonics have several advantages over numeric IDs, they are easier to remember, are simpler to exchange verbally, they prevent typing errors and may even provide a bit of humor.

2.2.12 Detecting duplicate uploads

We expect to deal with large datasets, uploading the same dataset twice can be a waste of time for users and tools. At the same time, we always encrypt the data with different initialization vectors and encryption key, making it impossible to detect duplicate datasets once they are stored encrypted. This is why we implemented a two-step hashing process during upload. We hash the original data of each chunk we receive using SHA-256, and we create a dataset hash by concatenating all the chunk hashes in order and hashing the hashes again. The hash behavior also is implemented separately in the CLI with the command `dabih hash <file-path>`. If a client starts an upload it may send the hash of the first chunk that will be uploaded, the dabih server will check if such a chunk exists for the user, and if it does, respond with the full hash of the matching dataset. The client can then hash the local file fully and check if it matches. If the hash matches the client may then cancel the upload, the default behavior for the currently implemented clients, but it can also continue uploading. This scheme allows us to detect duplicates, at almost no additional compute cost in the normal case.

2.2.13 Restarting uploads that were interrupted

The hashing algorithm described in 2.2.12 also allows us to restart interrupted uploads.

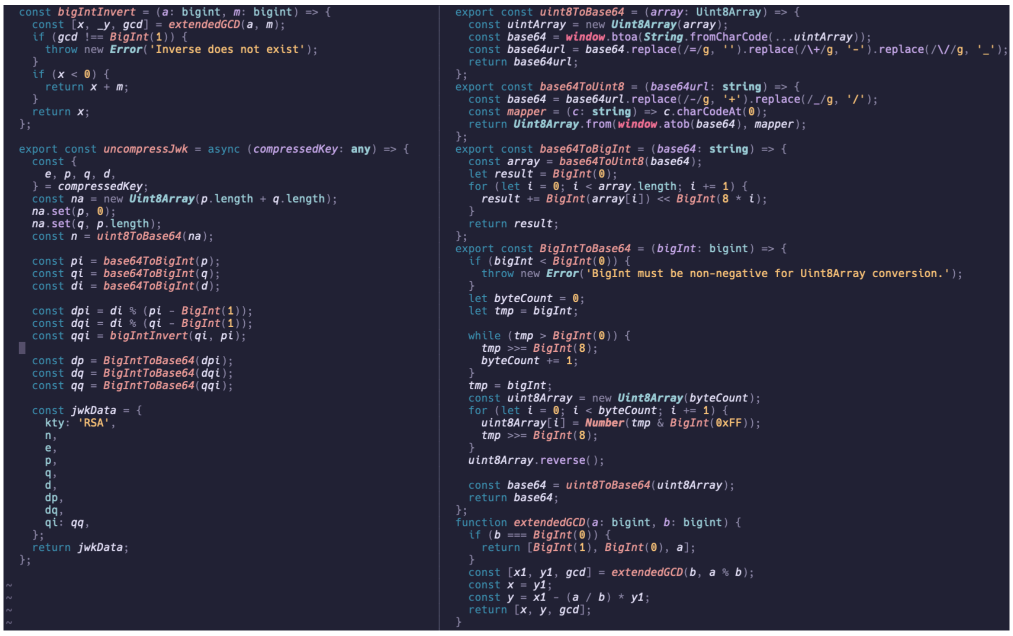
The client may ask the server for incomplete uploads and, if one exists, will receive a filename, and a list of already complete chunks and their hashes. This allows the client to load the file again, skipping the upload of the completed chunks, but reading and hashing them instead to ensure the data is identical. The dabih web client and the CLI implement this feature.

2.2.14 Reducing RSA-4096 private key size

Our selection of cryptography algorithms is limited by what is available in the browsers we target and by what the algorithms supported use cases are. The only valid algorithm was RSA, with the minimum modulus length of 2048, see [23]. We wanted to set the recommended modulus length of 4096, see [29]. At the same time we wanted users to be able to print and scan their keys as QR Codes, as defined by ISO/IEC 18004:2015 [53]. The Web Crypto API [33] has functions for exporting keys. For RSA private keys the formats

PKCS#8 [59] and JSON Web Key (JWK) [57] are supported. Unfortunately both formats are impractical for this use case because the exported size of the key is too large. The default JWK output is too large for any QR Code, the PKCS#8 output produces about 2370 bytes of output, which would only fit into a `Version 40` QR Code with low error correction. We were unable to scan this type of code consistently with a computer webcam.

But the JWK Format gives us a more mathematical representation of the RSA private key, with the following values: `n` the RSA public modulus, `e` the RSA public exponent, `p` the smaller RSA secret prime, `q` the larger RSA secret prime ($p < q$), `d` the RSA secret exponent $d = e^{-1} \bmod (p-1)(q-1)$, `dp` $= d \bmod (p-1)$, `dq` $= d \bmod (q-1)$, `qi` the multiplicative inverse $qi = p^{-1} \bmod q$. To compress the key we can remove `n, dp, dq, qi` from the JWK and recalculate them when we re-import the key, see Figure 2.7. This results in a smaller, very usable QR Code, even with medium level error correction, as seen in Figure 2.9.



```
const bigIntInvert = (a: bigint, m: bigint) => {
  const [x, _y, gcd] = extendedGCD(a, m);
  if (gcd !== BigInt(1)) {
    throw new Error('Inverse does not exist');
  }
  if (x < 0) {
    return x + m;
  }
  return x;
};

export const uncompressJwk = async (compressedKey: any) => {
  const {
    e, p, q, d,
  } = compressedKey;
  const na = new Uint8Array(p.length + q.length);
  na.set(p, 0);
  na.set(q, p.length);
  const n = uint8ToBase64(na);

  const pi = base64ToBigInt(p);
  const qi = base64ToBigInt(q);
  const di = base64ToBigInt(d);

  const dpi = di % (pi - BigInt(1));
  const dqi = di % (qi - BigInt(1));
  const qqi = bigIntInvert(qi, pi);

  const dp = BigIntToBase64(dpi);
  const dq = BigIntToBase64(dqi);
  const qq = BigIntToBase64(qqi);

  const jwkData = {
    kty: 'RSA',
    n,
    e,
    p,
    q,
    d,
    dp,
    dq,
    qi: qq,
  };
  return jwkData;
};
~
~
~
~
```

```
export const uint8ToBase64 = (array: Uint8Array) => {
  const uintArray = new Uint8Array(array);
  const base64 = window.btoa(String.fromCharCode(...uintArray));
  const base64url = base64.replace(/=/g, '').replace(/\+/g, '-').replace(/\//g, '_');
  return base64url;
};
export const base64ToUint8 = (base64url: string) => {
  const base64 = base64url.replace(/-/g, '+').replace(/_/g, '/');
  const mapper = (c: string) => c.charCodeAt(0);
  return Uint8Array.from(window.atob(base64), mapper);
};
export const base64ToBigInt = (base64: string) => {
  const array = base64ToUint8(base64);
  let result = BigInt(0);
  for (let i = 0; i < array.length; i += 1) {
    result += BigInt(array[i]) << BigInt(8 * i);
  }
  return result;
};
export const BigIntToBase64 = (bigInt: bigint) => {
  if (bigInt < BigInt(0)) {
    throw new Error('BigInt must be non-negative for Uint8Array conversion.');
  }
  let byteCount = 0;
  let tmp = bigInt;

  while (tmp > BigInt(0)) {
    tmp >>= BigInt(8);
    byteCount += 1;
  }
  tmp = bigInt;
  const uint8Array = new Uint8Array(byteCount);
  for (let i = 0; i < byteCount; i += 1) {
    uint8Array[i] = Number(tmp & BigInt(0xFF));
    tmp >>= BigInt(8);
  }
  uint8Array.reverse();

  const base64 = uint8ToBase64(uint8Array);
  return base64;
};
function extendedGCD(a: bigint, b: bigint) {
  if (b === BigInt(0)) {
    return [BigInt(1), BigInt(0), a];
  }
  const [x1, y1, gcd] = extendedGCD(b, a % b);
  const x = y1;
  const y = x1 - (a / b) * y1;
  return [x, y, gcd];
}
```

Figure 2.7: The key decompression algorithm, part of the dabih client [40]. It first converts the base64 encoded numbers into the `BigInt` type, able to represent numeric values too large for the `number` type. It then does the calculations described in 2.2.14 and converts the numbers back into base64 strings.

2.3 Results

Dabih is available open-source on GitHub [40]. All the features described in methods are implemented by the server and web-client in the repository. The main application flows of *uploading*, *sharing* and *downloading* are summarized in Figure 2.8. The full code can be audited before using dabih to secure data.

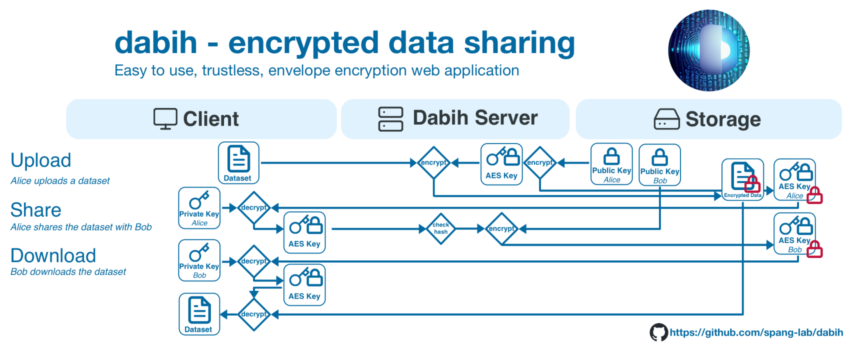


Figure 2.8: **Schematic overview of the dabih application:** Data is **uploaded** to dabih and then encrypted in two stages, the encrypted data is written to disk. This data can then be securely **shared** with authorized recipients, by re-encrypting the symmetric key with the new users public key. **Downloading** is the reverse of the upload process, decrypting the data in two stages.

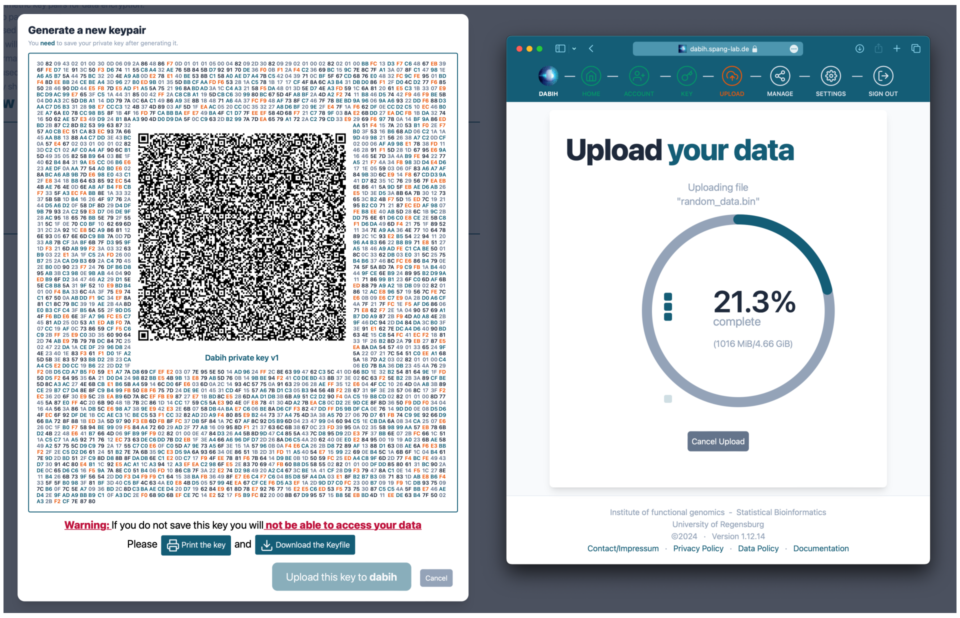
None of the security measures rely on secrecy of the code. Documentation for deployment is provided as part of the code repository. For ease of deployment we also provide a ready to use container [41] and example deployments for different environments. After deployment, dabih is directly available as an easy to use web-application. Most common administrative tasks are implemented as part of this web application, after configuring a list of users with *admin* privileges. Administrators can enable keys, delete datasets, and access activity logs, but they cannot access data from others.

2.3.1 Usability

While the full source code and all cryptographic details are available, a key design consideration for dabih is that users should never actively need to deal with the cryptosystem. After the initial setup users just upload, share and download data. In addition to their account users only need to manage their private key. We made this as simple as possible, allowing users to download their key as a file or by printing it as QR Code. We encrypt all data and do not offer a way of storing data unencrypted, as this would only create a risk of misuse. In almost all cases the upload speed to the dabih server is the limiting factor for performance, the encryption creates almost no performance cost. As dabih primary use is in the browser we depend on some browser API, but we only require the Web Cryptography API and the local storage for the client, these are supported by all the major browsers. The only notable browser we do not support is Internet Explorer, installed on older Windows computers.

2.3.2 Command line interface

For advanced use cases we provide an API reference as part of dabih, after generating an access token dabih can be used by any program through the web API. As part of the dabih source code we also provide a dabih command


Generate a new keypair

You **need** to save your private key after generating it.

Dabih private key v1

Warning: If you do not save this key you will **not be able to access your data**

Please Print the key and Download the Keyfile

Upload this key to **dabih** Cancel

Upload your data

Uploading file "random_data.bin"

21.3% complete

(1016 MiB/4.66 GiB)

Cancel Upload

Institute of functional genomics · Statistical Bioinformatics
University of Regensburg
©2024 · Version 1.12.14
Contact/Impressum · Privacy Policy · Data Policy · Documentation

Figure 2.9: *Left side:* A dabih private key in a printable format. The key is encoded as a QR Code that can easily be read by a computer webcam. We use a special smaller format, see 2.2.14, to be able to fit the key into a single QR Code. The QR Code is encoded as text, to allow for easy copy and pasting of the data. The key is also printed out as text next to the QR Code as a redundancy measure, typing it in should never be required.
Right side: The dabih web client, currently uploading a large file. We show a clear progress indicator, can detect duplicate uploads and can resume from incomplete uploads.

line interface (CLI), written in Rust. This CLI can be compiled by users or downloaded from our releases page on GitHub. We provide a pre-compiled binary for all major operating systems. It implements all the major functions of the graphical dabih client but can be used in shell scripts or for other automation tasks. In the case of data upload we even implemented additional features, not possible in the browser, such as recursively searching the file-system, or zipping folders before upload.

2.3.3 Data ingestion application

As, by design, a private key is not required for uploading data we use this as a feature for data ingestion. A user generates a simple randomly generated upload token, that can be sent to others and enables them to upload data to their account. We offer a separate app, based on the dabih CLI 2.3.2. This app is deliberately kept simple and only implements uploading to upload links for data ingestion as described in 2.2.7. It is based on *tauri*[19] which allows us to build an app for all major operating systems that calls our rust code in the dabih CLI package. It is available on our releases page on GitHub [40].

2.3.4 Collaboration and evaluation

dabih was developed based on regularly requirements, and follows recommendations from the federal office of information security [29]. As part of the TRR305 it was evaluated by both the FAU Competence Center for Research Data and Information (CDI) and the Regensburg University computer center. Minor changes, suggested by them, were implemented.

2.4 Discussion

We developed dabih as a more secure alternative to classical self-hosted file storage platforms like *Nextcloud* [31] and *Seafile* [65], while maintaining their user-friendliness. Although these platforms provide encryption, they typically use symmetric encryption with a password-derived key. dabih enhances security by introducing asymmetric key infrastructure. Nevertheless, it is still designed for users who may not be familiar with command-line tools. Its deployment, along with authentication and authorization processes, are as straightforward as those of the aforementioned services. Dabih focuses exclusively on the core feature of file storage, leading to a simple and easy to audit system. In the context of securing genomic data dabih's cryptography is similar in concept to the *Crypt4GH*[90] software, but we also include key management and, through different choice of algorithms, we not require users install software. In clinical settings, dabih can interoperate with other data management software, through its platform independent and well documented API. While it is not a replacement for clinic information systems, it offers a viable solution for managing sensitive research data. One of the key advantages of dabih is that it does not require custom software installations on clinic computers. Instead, all functionalities are supported by standard web browsers, which are standardized and maintained by multiple large corporations. The browser-based cryptosystem is arguably more secure than many other libraries. This is because they are widely used, ensuring that any vulnerabilities are quickly addressed. Additionally, each clinic has a strong incentive to keep their employees' browsers updated for security purposes. Differently from custom data management software, which require deployment and maintenance on client devices, dabih can be provided for free, with only server maintenance needed. While dabih adheres to best practices for data security, its overall security heavily relies on proper user behavior. This is particularly crucial as users may not be entirely familiar with asymmetric cryptography. It is imperative for data owners to maintain the confidentiality of their private keys and ensure they are not lost. Additionally, dabih's effectiveness depends upon the reliability of its storage backend. The encrypted data must be regularly backed up, and the file system should prevent data corruption, for instance, through the use of checksums. Due to the nature of the encryption algorithm, even a single bit flip in the encrypted data can corrupt the entire file. To mitigate this, we generate a checksum for every 2 MiB chunk of stored data. This is primarily for data validation and should be reserved for emergency recovery purposes.

2.5 Conclusion

We propose dabih as a viable and secure alternative for on-premise file storage and sharing solutions, while maintaining an equivalent level of user-friendliness. Leveraging the ubiquity of web browsers, which are installed on virtually every computer, dabih facilitates accessible and secure cryptographic operations for a broad user base.

Cryptographic Summary

- Symmetric Algorithm: 256-bit Advanced Encryption Standard with Cipher Block Chaining, AES-256-CBC, specified in NIST SP800-38A[70].
- Asymmetric Algorithm: 4096 bit Rivest Shamir Adleman (RSA) with Optimal Asymmetric Encryption Padding (OAEP), specified in RFC3447[83].
- Hashing Algorithm: 256-bit SHA (SHA-256) as specified in FIPS 180-4 [71].
- During file upload the dataset is processed in memory and not written to disk.
- When the upload starts the server generates a cryptographically strong pseudo-random AES-256-CBC Key k (24 Bytes)
- The client creates "chunks", sequential byte buffers of the data, each with size 2 MiB.
- For each chunk we again generate a cryptographically strong pseudo-random initialization vector (iv)
- The raw chunk data is hashed using SHA-256 and then encrypted using the AES key k with the initialization vector iv
- We then create a crc32 checksum of the encrypted chunk.
- This encrypted chunk is then written to the file system and the iv, hash and checksum written to the database.
- All the asymmetric keys are RSA key-pairs, with at least 4096 bits.
- We only encrypt the 24 Byte AES Key k using RSA.
- To prevent key exchange attacks all keys are fingerprinted using an SHA-256 hash.

Table 2.1: Detailed description of cryptographic algorithms used by dabih

3 Data Management for Next-Generation Sequencing

As described in chapter 2, we have a secure way of storing large datasets. The design of this storage was deliberately kept simple for security and auditing purposes. While dabih is well suited to work with biomedical datasets, especially human genome data, its design is completely data agnostic. dabih essentially provides a unique ID for datasets uploaded, and these IDs may be connected to metadata which is specific to the use and content of the data. Metadata can include details about the origin, structure, content, and context of the primary data, allowing users to understand and use the data effectively. For the more specific use case of RNA sequencing we created a web tool for managing the metadata.

3.1 Introduction

Next-Generation Sequencing (NGS) it the technology that enables the rapid reading of DNA and RNA sequences. It gives researchers the ability to decipher the genetic code with unprecedented speed and accuracy.

Over the past few decades the field of genomics has changed dramatically. Traditional and expensive techniques such as Sanger sequencing [85] have been largely replaced by more efficient next-generation technologies such as RNA-seq, ChIP-seq, or Nanopore Sequencing.

In cancer research, RNA-seq provides a snapshot of the transcriptome, the complete set of RNA molecules within a cell or tissue. RNA-seq is a cornerstone for our research consortium on metastatic organ colonization in cancer, TRR305 [101]. The process begins with isolation of RNA, followed by a conversion to complementary DNA (cDNA), library preparation, and high-throughput sequencing. The resulting data is then typically mapped to a genome and gene expression levels are quantified. For responsible research, this whole process, from biological sample to scientific results must be well-designed, well documented and reproducible. Each RNA-seq experiment, will generate several gigabytes of raw data in the FASTQ Format, and both wet-lab researchers and bioinformaticians contribute and document various kinds of metadata on the experiment. The amount of data is continuously increasing, especially as the technology becomes cheaper, and proper data management is essential.

3.2 Methods

Storing metadata for NGS is essential for organizing and interpreting the vast amount of information generated during sequencing experiments. Metadata encompasses information about the experimental design, sample characteristics, and processing steps, providing context for the raw sequencing data. Properly managed metadata is crucial for reproducibility, data sharing, and downstream analysis.

3.2.1 Design principles

Our data management strives to achieve the FAIR [104](findable, accessible, interoperable, and reusable) principles. FAIR describes concise, domain-independent principles for both data and metadata.

- **F**indable: Metadata and data should be easy to find for both humans and computers.
- **A**ccessible: Once the data is found, users need to know how it can be accessed, including authentication and authorization.
- **I**nteroperable: The data needs to be integrated with other data, and must be readable by applications or pipelines.
- **R**eusable: Data must be well described, so it can be replicated or used in different settings.

3.2.2 Design

Metadata is typically collected as a locally stored matrix, e.g. an Excel sheet with a list of *samples* as rows, and a set of *properties* as columns, with different matrices for different projects of contexts. This design is familiar to researchers but has a few disadvantages.

In this design metadata is not based on matrices, but instead on a continuous timeline of metadata *events*. Events track all the changes to metadata, with a sample reference, a property reference and a new value. Each event has a time-stamp and events are never deleted, newer events simply overwrite older events.

This requires a centralized design, instead of distributed, locally stored matrices. The key advantage is that there is a single source of truth for the metadata, metadata cannot diverge if there are changes from two different users to their local versions. But this design creates a set of new challenges for data security, redundancy and usability that need to be addressed.

3.2.3 Definitions

- **Sample:** Samples are single entries of data, and in most cases will correspond to individual biological specimens. Each sample represents a distinct biological condition or experiment group.

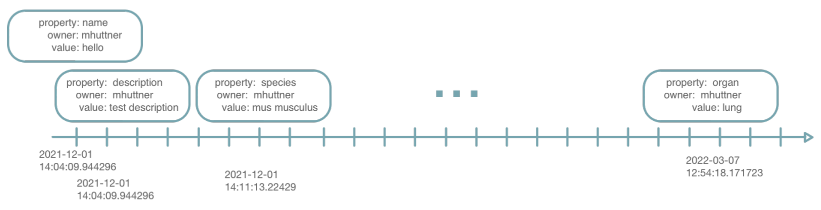


Figure 3.1: Metadata event timeline: events are never deleted, and have exact time-stamps. This allows us to recover the metadata at any time point.

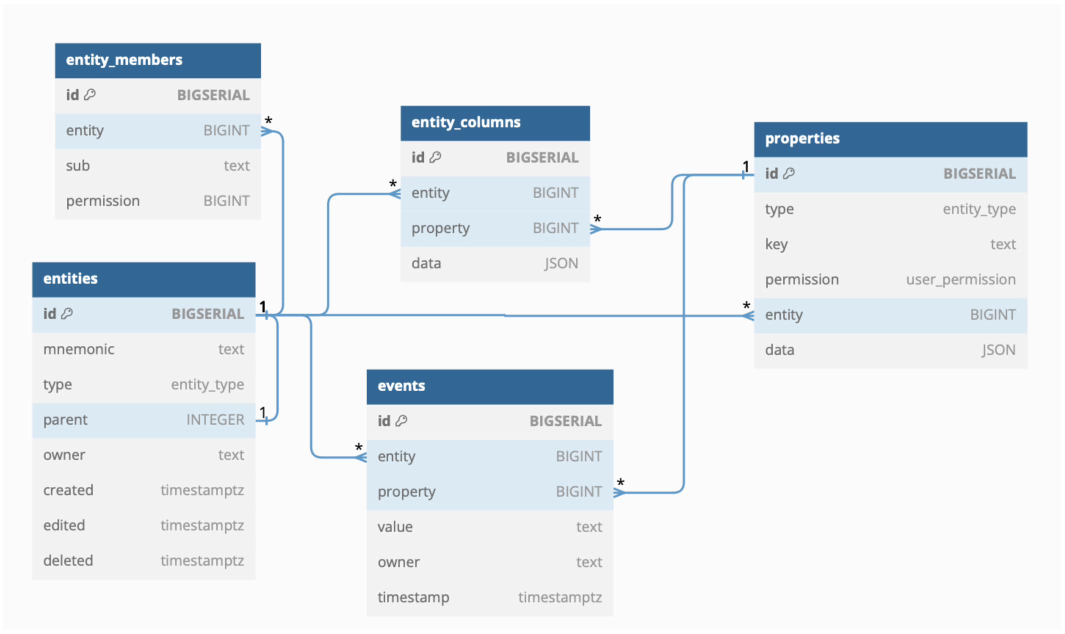
- **Property:** Properties describe the different kinds of metadata attached to a sample, common properties are: Sample identifier, experimental condition, tissue type, sequencing method, species, etc.
- **Project:** Projects are for organizing samples and for defining access to samples. Projects contain samples, and projects have members with various permissions. Projects describe the samples contained in them, and they are findable by all users.
- **Template:** Templates are logical collections of properties. E.g. all the properties required to submit a dataset based on EGA standards [24].
- **Entity:** Both samples and projects are entities. Entities can be related to other entities and all entities may have attached metadata.
- **Value:** Values are the actual metadata, e.g a property *species* may have the values *homo sapiens* or *mus musculus*.
- **Event:** Events represent changes to the metadata, an event references an entity, a property and a new value. Each event also has a time-stamp and an owner.

3.2.4 Structure

Metadata collection is implemented as a server-client system, see 1.7, complementary to the data storage (See chapter 2). The server provides a web API, receives and sends and converts the data, and talks to the database.

3.2.5 Computational access

Metadata is automatically available in a machine-readable format from the servers JSON API, see 1.7.2. Independent of if the metadata samples already have a unique identifier, each sample is automatically assigned a unique mnemonic [3.1F1]. The mnemonic ID system is the same as for dabih, see 2.2.11. We use these mnemonics for creating deep links to the data. Deep links are full hyperlinks that point directly to the sample. With the deep links



entity_members	
id	BIGSERIAL
entity	BIGINT
sub	text
permission	BIGINT

entity_columns	
id	BIGSERIAL
entity	BIGINT
property	BIGINT
data	JSON

properties	
id	BIGSERIAL
type	entity_type
key	text
permission	user_permission
entity	BIGINT
data	JSON

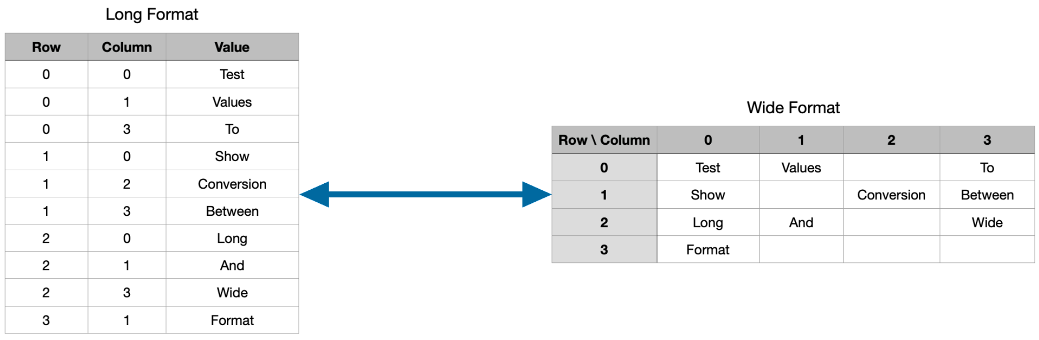
entities	
id	BIGSERIAL
mnemonic	text
type	entity_type
parent	INTEGER
owner	text
created	timestamptz
edited	timestamptz
deleted	timestamptz

events	
id	BIGSERIAL
entity	BIGINT
property	BIGINT
value	text
owner	text
timestamp	timestamptz

Figure 3.2: Database structure for the metadata. Metadata is stored as long list of events.

we also generate a QR Code for each sample, that may be used as a sticker or printout attached to the physical sample. The API is based on HTTP[3.1A1.1] and includes authentication and authorization [3.1A1.2]. The metadata API is independent of the dabih API for data [3.1A2].

3.2.6 Format conversion



Long Format

Row	Column	Value
0	0	Test
0	1	Values
0	3	To
1	0	Show
1	2	Conversion
1	3	Between
2	0	Long
2	1	And
2	3	Wide
3	1	Format

Wide Format

Row \ Column	0	1	2	3
0	Test	Values		To
1	Show		Conversion	Between
2	Long	And		Wide
3	Format			

Figure 3.3: Schematic for conversion from a long data format to a sparse wide matrix. This conversion is used for converting between the metadata timeline and a familiar matrix format.

Metadata is stored as a timeline, but presented to users in a matrix format. This requires a conversion process. This process is show in Figure 3.3 works like this:

1. For a project, find all the sample IDs belonging to that projects.
2. Fetch all events that are related to one of these sample IDs, sorted by their time-stamp.
3. Create a new empty matrix with samples as Rows and Properties as columns.
4. For each event find the corresponding row and column in the matrix and fill in the event value.
5. If a matrix field is already filled by a previous event, overwrite the value with the newer events value.
6. This results in the full matrix.

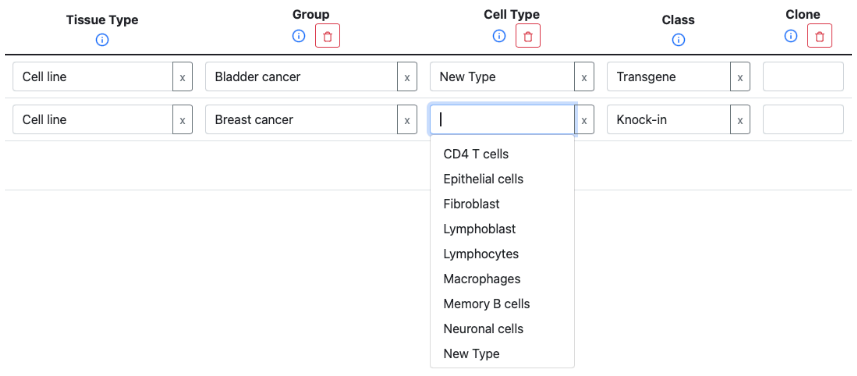
This process is lossy, because we overwrite older values, but this is by design. The latest state of the data is most relevant to users. We can adapt this conversion process to recover older data, see 3.2.10. This conversion process is only done in one direction, from events to matrix, changes to the data can be directly recorded as new events. With our SQL based database (see 1.7.1) we can define indices that make this conversion process fast. Even for large matrices the whole conversion takes less than 100ms.

3.2.7 Metadata templates

While the properties that are recorded are set by users and are project specific, we provide lists of properties as templates for new user projects. These templates serve as a starting point and may be adapted for each project. They are based on standards set by the GHGA [96] and the EGA [24], and make sure users capture and record all the data required for later submission and publication of data [3.1R1.3]. New templates for different use cases can be added easily, and users select the approritate template on project creation.

3.2.8 Validation and normalization

The properties we define also control the vocabulary used, and prevent typos. E.g. for a property like *species*, it is not clear which words should be used. *Human*, *human* or *homo sapiens* could be used interchangeably, but later automated computer analysis may treat them as different. As part of the properties, where reasonable, we provide a description, a set of default options and a validation function. e.g. for *species* the description would be *"Latin name for the organism"*, with examples *homo sapiens*, *mus musculus*, *rattus norvegicus* and *danio rerio*. These options are presented as part of the UI when entering the data, see Figure 3.4. Users are not limited to the suggested values, they may enter anything and the values entered for other samples are also suggested when entering data. This is an effective way to prevent typos and create a uniform vocabulary [3.1I1, I2], while at the same time speeding up the users work. We use regular expressions, patterns for matching strings, to validate input from users. Invalid input will be highlighted in red with a clear error message. Validation only provides information to the user, we do not force compliance or prevent input, users stay in control of their data. We provide a graphical interface to administrators for creating and editing properties.



Tissue Type	Group	Cell Type	Class	Clone
Cell line	Bladder cancer	New Type	Transgene	
Cell line	Breast cancer		Knock-in	

Figure 3.4: Metadata Auto completion: Metadata values are automatically suggested based on the metadata template and based on previously entered values, this prevents typos from retyping and makes sure metadata uses uniform vocabulary.

3.2.9 Data integration

As already described, large dataset are stored encrypted in dabih (see chapter 2), because each dataset has a unique ID, assigned by dabih, it is sufficient to simply track the ID as part of the metadata [3.1I3, F3]. Storage related metadata, such as the data size, file name, file hash or storage location is tracked by dabih and does not need to be repeated in the metadata.

3.2.10 Reproducibility and time travel

In the case of accidental removal of data or other errors we can adapt the conversion process described in 3.2.6 to recover metadata, or undo unintended changes. If, instead of fetching all events, we only fetch events up to a certain time point, and then convert, we end up with a version of the data at that time point. This way data is never deleted but only overwritten. With the computational access 3.2.5 from the JSON API, advanced users can fetch the event data and arbitrarily filter it e.g. with open tools such as jq [21]. This is great for data security but may also have negative consequences. Personalized data that was entered accidentally cannot be easily deleted, and must be removed by an administrator.

3.2.11 Security

While the security of the data is not as critical as the raw datasets, because personally identifiable information should never be part of the metadata, it is nonetheless important. As already described in 3.2.10 data is already protected from accidental deletion. We also carefully manage access based on projects. Each project has an owner who may grant read or write access to other users. Because the data is recoverable, write access may be granted liberally. In order to enable findability of data, project names and descriptions are available and searchable by all users [3.1F4].

3.2.12 Redundancy

As described in 1.7.1, on our infrastructure all the metadata is stored in multiple databases with automatic fail-over. Regular backups are created and user may optionally keep local copies of the data. We rely on the database to provide the data redundancy, where there are multiple well established options.

3.3 Results

Data storage in dabih, together with the metadata collection create a solid foundation for NGS data analysis, based on FAIR principles [104]. These tools are useful for ensuring data is handled according to best practices while at the same time helping researchers be more productive. We abstract away complexities such as encryption, data validation, backups or publication, enabling researchers to focus on their research. Open protocols and formats allow for easy integration with other tools and pipelines. We enforce ontologies and a controlled vocabulary, improving the quality of the data, without extra effort from researchers.

Dabih is already published, see 2.3, the metadata collection code will be published in the future building on top of dabih.

3.3.1 Data workflow

Our tools are in active use as part of the TRR305 [101], for studying the mechanisms of metastatic organ colonization in cancer. The data workflow can be seen in Figure 3.5.

3.4 Discussion

Data and metadata management is important not just for NGS workflows but for all projects with large datasets. Software faces an important trade-off: between being usable and generalized enough for a wide variety of datasets and use cases, and with providing the specialized solutions and tools users require. While turn-key data management solutions like rucio [6] or invenio [16] scale to huge data sizes required e.g. for high energy physics, them being application independent will require custom code for adaption or high effort from users for each use case. With rule-base frameworks such as iRODS [82] integrations like the required data encrypted could be implemented. With the metalnx [74] plugin iRODS even offers web based interface similar to our tools. But integrating other data models, not based on files, like the one we use for metadata, would still be difficult to impossible. Our tools provide their full functionality through open JSON APIs. This still enables integration with these rule based frameworks or other data management tools, if required. With our clear focus on NGS data analysis we can provide additional value to users, provide context specific data and integrate with domain specific tools and standards, like the ones from EGA [24]. This comes at the cost of being less domain independent. With our design we only create this domain specificity for tracking metadata and keep data storage domain and data independent.

3.5 Outlook

Work on the data management continues in the TRR305, in collaboration with bioinformaticians from Erlangen, Fulvia Ferrazzi and Renato Liguori. While dabih and the metadata collection are well integrated when running on our cluster we intend to publish the software with minimal dependencies. We will further develop this integration, and may offer an easily deployable full data management solution based on these tool in the future. Dabih also has more extensive, authentication provider independent, API access through custom access tokens. We will enable this also for the metadata, with a fully documented API similar to dabih. With this shared authentication system, and cryptographic keys already required by dabih, we plan to also offer optional encryption of the metadata, further increasing data security. Pamly, the application we created for AI pathology described in the next chapter already implements automatic user-triggered pipelines for data. With unified API access to both data and metadata it is also possible to automatically trigger basic and computationally cheap NGS pipelines.

FAIR principles
Findable

- **F1.** (Meta)data are assigned a globally unique and persistent identifier
- **F2.** Data are described with rich metadata (defined by R1 below)
- **F3.** Metadata clearly and explicitly include the identifier of the data they describe
- **F4.** (Meta)data are registered or indexed in a searchable resource

Accessible

- **A1.** (Meta)data are retrievable by their identifier using a standardized communications protocol
- **A1.1** The protocol is open, free, and universally implementable
- **A1.2** The protocol allows for an authentication and authorization procedure, where necessary
- **A2.** Metadata is accessible, even when the data is no longer available

Interoperable

- **I1.** (Meta)data use a formal, accessible, shared, and broadly applicable language for knowledge representation.
- **I2.** (Meta)data use vocabularies that follow FAIR principles
- **I3.** (Meta)data include qualified references to other (meta)data

Reusable

- **R1.** (Meta)data are richly described with a plurality of accurate and relevant attributes
- **R1.1.** (Meta)data are released with a clear and accessible data usage license
- **R1.2.** (Meta)data are associated with detailed provenance
- **R1.3.** (Meta)data meet domain-relevant community standards

Table 3.1: The full numbered fair principles from the Go Fair web page [105] based on the FAIR publication [104].

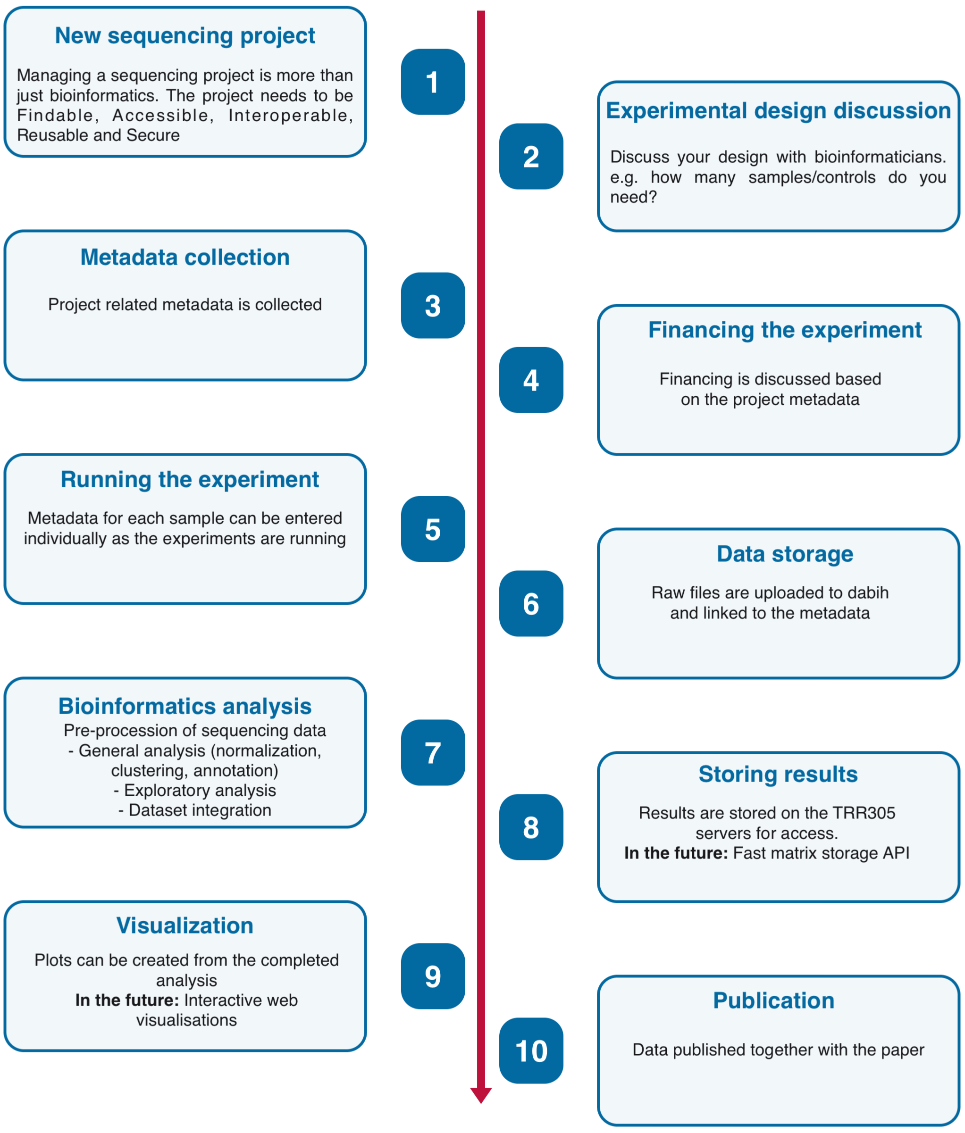

New sequencing project
Managing a sequencing project is more than just bioinformatics. The project needs to be Findable, Accessible, Interoperable, Reusable and Secure
1
2
Experimental design discussion
Discuss your design with bioinformaticians. e.g. how many samples/controls do you need?
Metadata collection
Project related metadata is collected
3
4
Financing the experiment
Financing is discussed based on the project metadata
Running the experiment
Metadata for each sample can be entered individually as the experiments are running
5
6
Data storage
Raw files are uploaded to dabih and linked to the metadata
Bioinformatics analysis
Pre-procession of sequencing data
- General analysis (normalization, clustering, annotation)
- Exploratory analysis
- Dataset integration
7
8
Storing results
Results are stored on the TRR305 servers for access.
In the future: Fast matrix storage API
Visualization
Plots can be created from the completed analysis
In the future: Interactive web visualisations
9
10
Publication
Data published together with the paper

Figure 3.5: Full NGS workflow overview from the TRR305

4 pamly - digital pathology platform

4.1 Introduction

Histopathology is the practice of diagnosing tissue samples from patients. These tissue samples are cut into thin slices and stained using immunohistochemistry stain. Pathology used to be fully analog process from tissue sample to diagnosis under a microscope. New ways of image analysis created the field of digital pathology as a powerful tool to augment the practice of diagnosing diseases. The recent advances in AI image analysis, and the ability to download pretrained models ready for adaption created the now large and diverse field of AI image analysis.

Digital pathology involves the application of advanced algorithms and machine learning techniques to analyze high-resolution images of tissue slides, called whole slide images (WSIs).

These algorithms can process vast amounts of data and detect patterns, anomalies, and changes within the tissue. They provide pathologists with valuable support in diagnosis.

Unfortunately there a still major challenges in digital pathology, mostly at the interface between the pathologists and the AI researchers. As there is no universal format for WSIs, and most scanner vendors define their own proprietary image formats, analysis tools and slide viewers. To get the large amounts of data required for training effective AI models interoperability between data formats from different pathologists is required. Furthermore, AI results on these large dataset need to be visualized properly in order to be useful for pathologists.

4.2 Materials and methods

We developed *pamly*, an interactive web application, as a support tool for histopathology. The structure is similar to dabih, see chapter 2, with an API server and an interactive web client.

4.2.1 Histopathology process

First a tissue sample is obtained from a patient, with a biopsy. Most of our tools do not depend on a specific type of tissue, but the tissue samples we use are mostly whole lymph nodes, or needle biopsies of lymph nodes. The tissue is then properly dissected and embedded in paraffin wax. From the wax block multiple thin slices are cut and put onto glass slides. The slices are then stained using different immunohistochemistry markers, such as hematoxylin and eosin (H&E), CD20 or CD68. The "analog" pathology process continues from here, where a pathologist diagnoses the patient by viewing the slides under the microscope. In our case the slides are scanned using a high-resolution scanner. This creates whole slide images (WSIs).

4.2.2 Data

For a typical tissue area of about $2cm^2$ the scanned image file will have a size of about 2 GiB. Reading this scanned image is not easy, as most commercial scanners only create a proprietary format. This format is only meant to be read by special software sold by the manufacturer, e.g. a Hamamatsu Scanner creates *.ndpi* files, for their *NDP®.view2*[35] software.

4.2.3 Upload

With our work these slides can be uploaded over the web (or a local network) to the pamly server. All the files are split by the client into chunks (typically 2 MiB) which are reassembled by the server. Upload features are similar to dabih, with duplicate detection, see2.2.12 and the ability to resume incomplete uploads, see 2.2.13. Alternatively pamly also supports reading the data directly from the file system of the server, or dabih data ingestion can be used, see 2.2.7.

4.2.4 Hashing

As slide processing involves significant computational resources, we perform a 256-bit SHA hash computation [71] for every uploaded file, storing the result in the database. When a new file is uploaded, we compare its hash with existing hashes, and if a match is found, the file is skipped. Obtaining the same hash for two different files is practically impossible.

4.2.5 Preprocessing

We built a custom slide conversion tool based on the open-source library *openslide*[32] which can read different proprietary WSI formats from all popular scanners in use at pathology institutes.

As a target format we chose a *SQLite* [93] database file for each WSI. This database contains metadata and a set of tiles as a quad tree.

We pick a fixed tile size s and for each WSI we pad the image with white pixels, such that the scanned image is a square with size $s \cdot 2^n$, with the lowest possible n. We then split the slide into 4^n tiles with unique coordinates $(0, 0, n), (0, 1, n), \ldots, (0, 2^n - 1, n), (1, 0, n), \ldots, (2^n - 1, 2^n - 1, n)$. These tiles are then read one by one and stored in the database as JPEG encoded images. As part of this process two basic filters can be applied to each tile: a tile can be discarded immediately if it contains to many white pixels, or if a canny edge detector cannot find any edges. These filters can be configured by the user, and reliably filter out regions of the slide that do not contain tissue. All

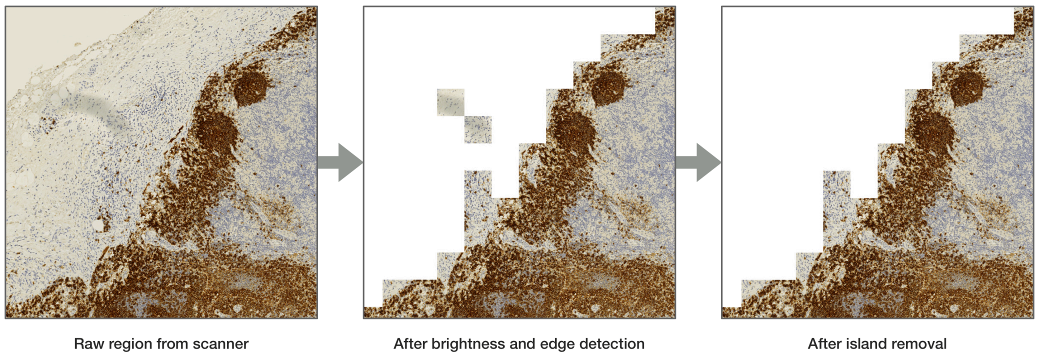


Figure 4.1: Visual example of the image preprocessing, showing which parts of the slide will be removed. The parameters for these filtering steps can be adjusted and the filtering can be disabled if required.

other tools will treat "missing" tiles in the database as background (typically white). In a further filtering step we detect *islands*: we cluster all remaining tiles into connected regions. Any two tiles are part of the same region if there is a path of non-background tiles connecting them. A valid tissue area will contain hundreds of connected tiles, so if a connected region is smaller than a certain size we remove it.

4.2.6 Scaling

After the filtering process we recursively compute lower resolution tiles, which simulate "zooming out".

We build a *quad tree* the following way:

- Start with the 4^n tiles in the database, created in 4.2.5, we refer to these tiles as level n tiles.
- Create 4^{n-1} new tiles by combining each 2x2 grid of tiles into a single image, e.g. the new tile $(0, 0, n-1)$ is the combination of tiles $(0, 0, n), (0, 1, n), (1, 0, n), (1, 1, n)$.
- These new tiles have size $2s \times 2s$, use a standard image algorithm to down-scale the image to size $s \times s$ again.
- These tiles now represent the level $n-1$.
- Repeat this process to create 4^{n-2} tiles to create the level $n-2$.
- Continue recursively creating levels until reaching level 0, containing a single tile.

See Figure 4.3 for a visual example of this process.

4.2.7 pamly pipeline

Depending on the size of the WSI and the server used, the conversion process will often take multiple minutes. We use a state machine to model and monitor this process, see Fig 4.2. Pamly can read these states for each slide, and can report the progress and any errors that may occur. This pipeline may also be extended, e.g. to directly run uploaded WSIs through a neural network for quality control or pre-diagnosis.

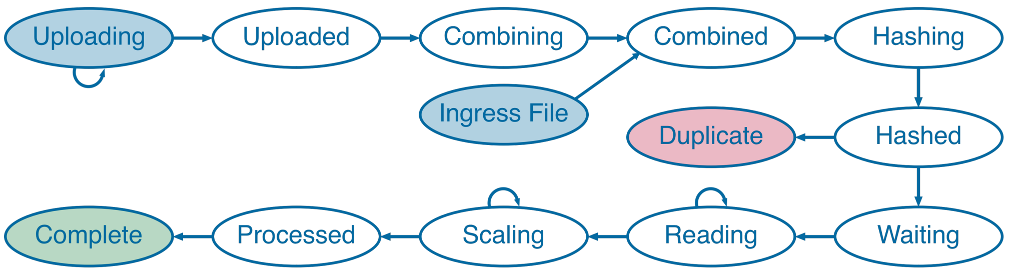


Figure 4.2: Pamly state machine for processing whole slide images. This process either starts with an upload of data, or with a new file in the ingress folder. If a file is uploaded, pamly will wait for all chunks to be uploaded, then combine the chunks into the original slide file. If a new file is added to the server the upload steps are skipped. In either case the new file is then hashed, checked against existing data and skipped if it was already processed. If the file is new, pamly starts the actual convert process, and reads progress updates for reading and scaling the slide. As soon as the convert process completes the slide's metadata is read into the database, and it is available for viewing.

4.2.8 Slide reading

We use this format because it allows for fast random access at any resolution, meaning the time it takes to read a rectangular region of the slide only scales with the size of region. This is easy to see as, given the tile size s, we can simply calculate all tiles that intersect the rectangular region we want to read, see Figure 4.3. For each rectangle with width w and height h at most $\lceil \frac{w}{s} \rceil \times \lceil \frac{h}{s} \rceil$ tiles intersect it and reading a single tile from the database is $\mathcal{O}(1)$.

4.2.9 Slide viewer

This fast random access is necessary for efficiently training neural networks on this image data, but also has an advantage for viewing the data as a human. When these slides are viewed under the microscope the pathologist will use different magnifications while moving the slide to view different regions. We can reproduce this process in a digital slide viewer in a very bandwidth efficient way, which allows live viewing of these slides over the internet. The key to this

is to recognize that when viewing a large tissue area, the human eye can no longer discern fine detail. We can instead show pre-computed lower resolution tiles. With higher magnification details become important, and we load high resolution tiles. At the same time, after zooming in, only a small part of the slide will be within view. We can detect scrolling in the viewer and load the tiles that will come into view just in time.
This behavior is implemented as part of the pamly web application. After a WSI is uploaded and processed, we store the resulting *sqlite* file and offer the slide data as a web API. This works the same way as the web clients for services such as OpenStreetMap or Google Maps, the web client requests the relevant tiles and combines them into a seamless scrollable map.

4.3 Results

Pamly is available open-source on GitHub [42]. Documentation for deployment is provided as part of the code repository. For ease of deployment we also provide ready to use containers and example deployments for different environments. After deployment, pamly is directly available as an easy to use web-application. WSIs can be added to pamly by either uploading the slide through the web UI or by moving the files into the pamly *ingress* folder.

4.3.1 Slide conversion tool

We designed pamly not only to be used as a web application. All the code for reading and converting the slides is written in rust [67] and published as a *crate* (rust's name for packages) on rust's main package repository *crates.io* [44].

This crate is both a library and a command line interface (CLI) that can be used directly. If users want to convert slides using the process described in 4.2.5 and 4.2.6 they need to install openslide [32] and install pamly with the *convert* feature. `cargo install pamly --features convert`.

Furthermore we provide a subset of the functions as bindings to the *python* programming language, published as a package on *PyPI* [43]. Most AI workflows are written in python and they can directly use pamly as a dependency or install it via `pip3 install pamly`. This also allows us to define consistent types and enumerations for the high-performance convert code written in Rust, the neural network package written in Python and the web application written in JavaScript. These types are defined in the Rust code, and then exported to python. This is enabled by the powerful Rust compiler and the bindings library PyO3 [81]. Only simple compiler annotations are required see Figure 4.6. The package is pre-built for all platforms using GitHub Actions, see 1.6.4.

4.4 Discussion

With our software we aim to make whole slide pathology images immediately usable by both pathologists and AI researchers, instead of relying on proprietary formats we use *SQLite* as a simple open and interoperable format that may be used from any other tool or programming language. As part of this we created an easy way to convert to this format and a simple way of both reading the data and viewing it. Our tool is built on top of openslide [32], with gives us a great interface for reading the slides. Closely related to our software is QuPath [5], with a more extensive feature set, but more focused on offline local work. Similarly to dabih, we use the power of the browser for our slide viewing and rendering, and do not require software installation. This is also the reason why pamly converts the slides into an SQLite based format first, users then do not need to compile the openslide library [32].

The methods used by pamly are well established, e.g. in SlideToolkit [72], but we provide them differently. We created a full pipeline that is automatically run after upload of the slides, results and potential problems are immediately visible to pathologists. We do not require users to be proficient at coding or scripting.

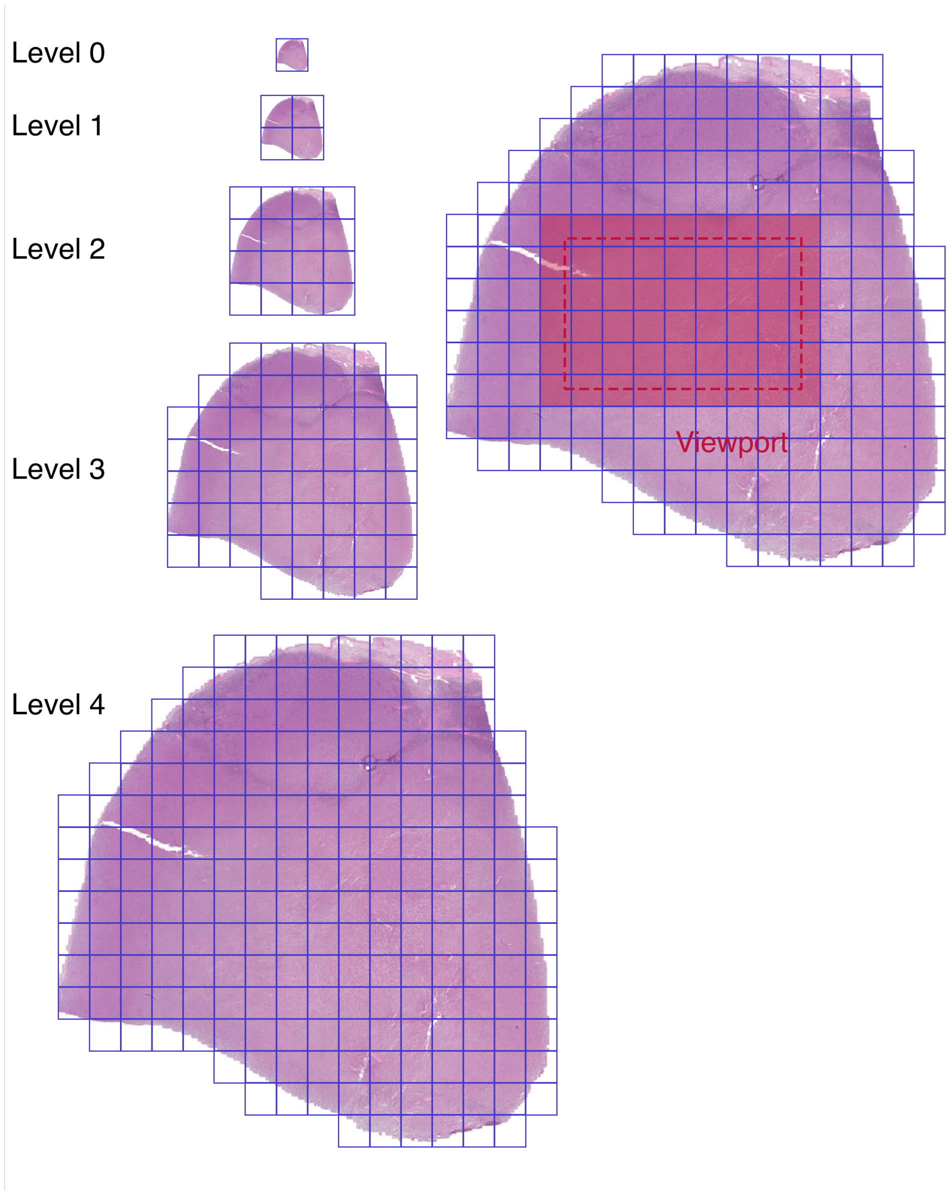
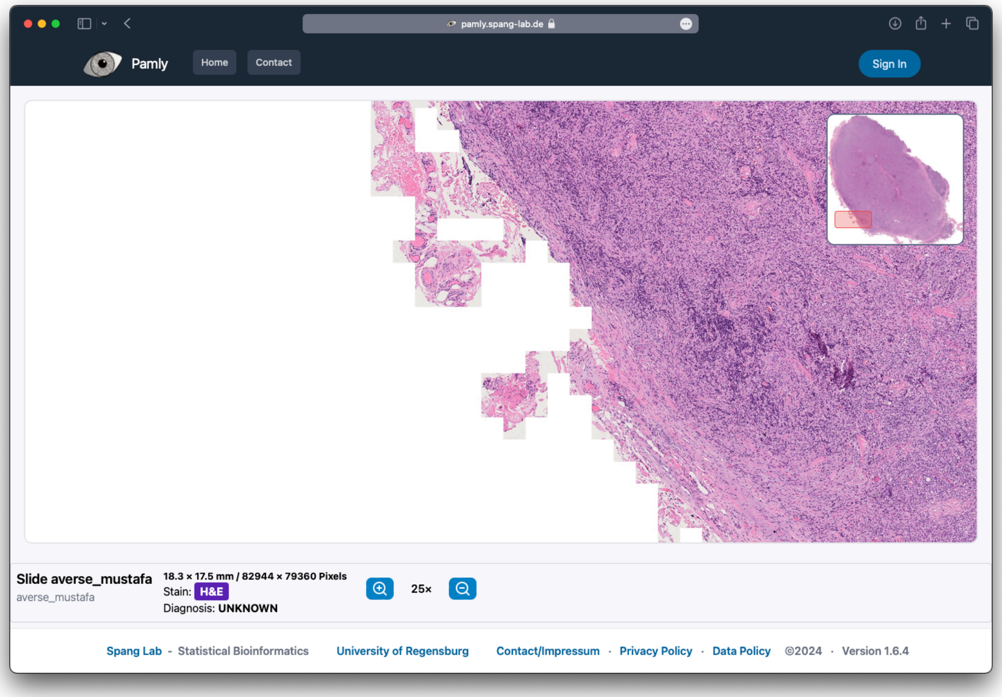
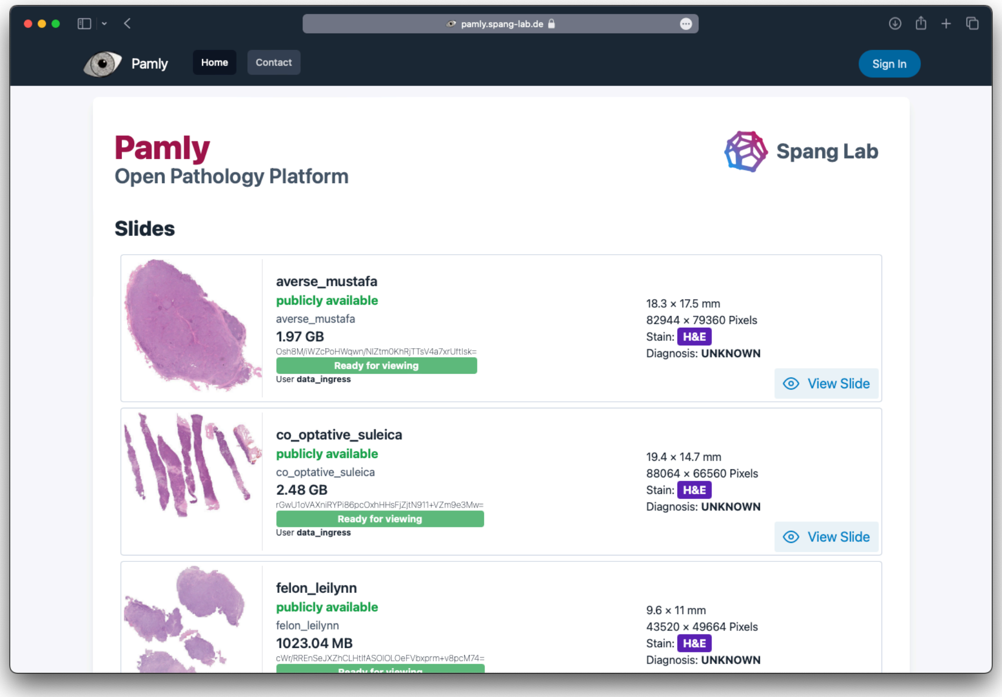


Figure 4.3: Left: How the slides are stored as a set of tiles, after scaling 4.2.6. We store the slide in its original resolution and a scaled copy for each level. The scale for each level l is 2^l in both width and height. Empty tiles are not stored. Right: How the storage format is used for viewing: the required tiles for a certain viewport are calculated and only those tiles are loaded on-demand.


pamly.spang-lab.de
Pamly
Home
Contact
Sign In
25×
Spang Lab - Statistical Bioinformatics
University of Regensburg
Contact/Impressum · Privacy Policy · Data Policy ©2024 · Version 1.6.4

Slide averse_mustafa
averse_mustafa
18.3 × 17.5 mm / 82944 × 79360 Pixels
Stain: H&E
Diagnosis: UNKNOWN

Figure 4.4: The web base viewer, it fetches tiles on-demand at different resolutions. It works similar to popular online map services, such as OpenStreetMap or Google Maps.


pamly.spang-lab.de

Pamly Home Contact Sign In

Pamly
Open Pathology Platform

Spang Lab

Slides

averse_mustafa
publicly available
averse_mustafa
1.97 GB
Osh8M/iWZcPoHWqwn/NIZtm0KhRjTTsV4a7xrUftlsk=
Ready for viewing
User **data_ingress**

18.3 × 17.5 mm
82944 × 79360 Pixels
Stain: H&E
Diagnosis: **UNKNOWN**

View Slide

co_optative_suleica
publicly available
co_optative_suleica
2.48 GB
rGwU1oVAXniRYPi86pcOxhHHsFjZjtN911+VZm9e3Mw=
Ready for viewing
User **data_ingress**

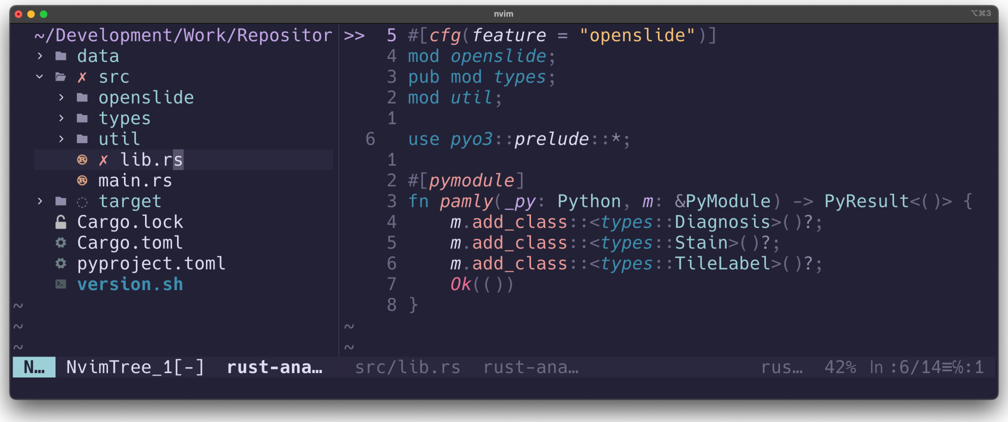
19.4 × 14.7 mm
88064 × 66560 Pixels
Stain: H&E
Diagnosis: **UNKNOWN**

View Slide

felon_leilynn
publicly available
felon_leilynn
1023.04 MB
cWr/RREnSeJXZhCLHtlfASOlOLOeFVbxprm+v8pcM74=

9.6 × 11 mm
43520 × 49664 Pixels
Stain: H&E
Diagnosis: **UNKNOWN**

Figure 4.5: The pamly web application, `https://pamly.spang-lab.de`. A few of the slides are publicly available.


```
~/Development/Work/Repositor
› data
⌄ ✗ src
  › openslide
  › types
  › util
    ✗ lib.rs
    main.rs
› target
  Cargo.lock
  Cargo.toml
  pyproject.toml
  version.sh
```

```rust
#[cfg(feature = "openslide")]
mod openslide;
pub mod types;
mod util;

use pyo3::prelude::*;

#[pymodule]
fn pamly(_py: Python, m: &PyModule) -> PyResult<()> {
    m.add_class::<types::Diagnosis>()?;
    m.add_class::<types::Stain>()?;
    m.add_class::<types::TileLabel>()?;
    Ok(())
}
```

```
N…  NvimTree_1[-]  rust-ana…  src/lib.rs  rust-ana…        rus…  42%  ln :6/14≡%:1
```

Figure 4.6: The rust python binding code. A simple rust compiler annotation from pyo3 [81] `#[pymodule]` exports rust functions to python. This prevents errors as custom types and enums are only defined once, in the rust code.

5 AI Pathology

In collaboration with Prof. Dr. Wolfram Klapper and Dr. Sarah Reinke from the institute of pathology of the Christian-Albrechts-University in Kiel a large dataset of whole-slide images (WSIs) was analyzed.

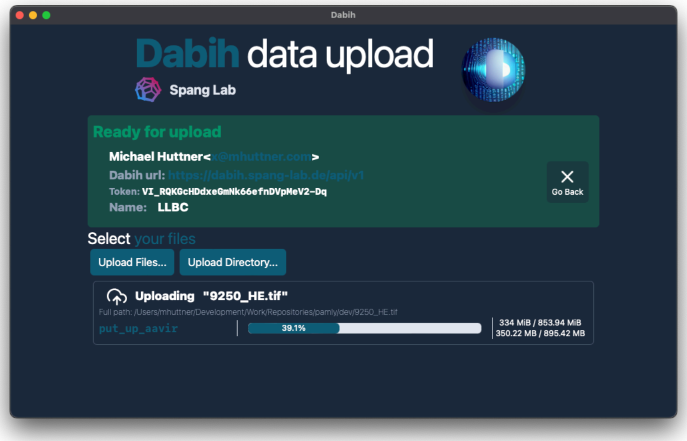
5.1 Introduction

After infrastructure challenges are solved and pathology data is acquired, the possibilities and applications of image analysis are endless [20]. Neural networks have proven to be adept at extracting and recognizing complex patterns from image data, making them invaluable tools for lymphoma diagnosis. While the theory on neural networks goes back to the 1960 [87], only the power of modern compute clusters enabled the creation of well performing models for image classification and segmentation. These models contain hundreds of layers to be trained, and are called deep neural networks. A key insight for deep learning was that these model generalize well: they can first be trained on a huge dataset of everyday images, like cars, trains or animals. They acquire general image recognition skills like edge detection that can then be transferred over to more specialized classification tasks [39]. This process is called transfer learning, and it is essential for training neural networks without the huge compute clusters of large entities like Google or Meta. Researchers can download published, pre-trained models, adapt them, and then fine-tune them to more specialized tasks with less data and compute resources. Many applications of deep learning on pathology are image segmentation tasks, e.g separating lymph node tissue from a breast cancer metastasis [22]. This is different in the case of lymphoma, because the lymph node does not contain foreign tissue, but is homogeneously affected by the cancer. This makes the task for lymphoma a classification and not a segmentation task, with fundamentally different base networks and methodology.

5.2 Methods

5.2.1 Collecting slides

As described in 2.2.7, we can use our data infrastructure for supporting the data transfer of the large WSIs from pathologies. While this would be supported by browsers, a large amount of data can be more easily uploaded with our custom app 2.3.3. With it users can recursively search the file system, upload whole folders and queue multiple uploads to run in the background. No account or cryptographic key is required by the uploader.


Dabih

Dabih data upload

Spang Lab

Ready for upload

Michael Huttner<x@mhuttner.com>

Dabih url: https://dabih.spang-lab.de/api/v1

Token: VI_RQKGcHDdxeGmNk66efnDVpMeV2-Dq

Name: LLBC

Select your files

Uploading "9250_HE.tif"

Full path: /Users/mhuttner/Development/Work/Repositories/pamly/dev/9250_HE.tif

put_up_aavir 39.1% 334 MiB / 853.94 MiB 350.22 MB / 895.42 MB

Figure 5.1: A whole slide image (WSI) being uploaded to dabih, using the dabih data ingestion tool described in 2.2.7

5.2.2 Dataset

The first dataset transferred contains 755 WSIs from 157 patients. All the images are cuts from a full lymph node. All the patients have been diagnosed by a pathologist, and received one of the following diagnoses:

- Chronic lymphocitic leukemia (CLL)
- Diffuse large B-cell lymphoma (DLBCL)
- Follicular lymphoma (FL)
- Hodgkin lymphoma (HL)
- Lymphadenitis (LTS)
- Mantle cell lymphoma (MCL)

For each patient there are multiple WSIs, with different immunohistochemistry stains. These stains are visible markers applied to the tissue, and they bind to specific antigens.

- **H&E:** Hematoxylin and Eosin is the most common staining method, and visualizes the general morphology the tissue.
- **CD20:** is a marker for B-lymphocytes, it is commonly used to identify B-cells in the tissue.
- **CD30:** is a marker for activated T-cells, especially CD4+ T cells.
- **CD68:** is a marker for macrophages and monocytes.
- **CD3:** is a marker for both T-cells and natural killer cells.

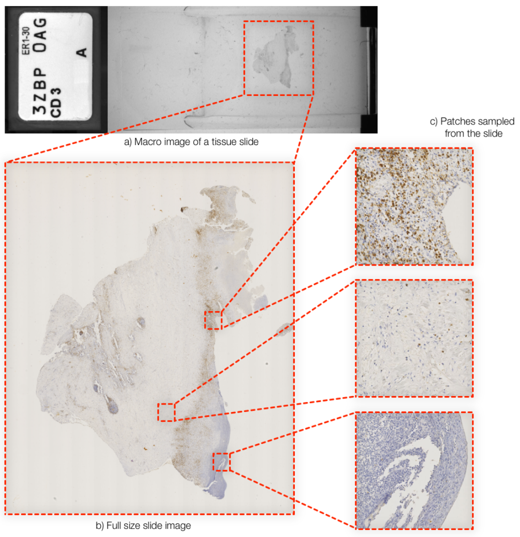

ER1-30
3ZBP OAG
CD3
A
a) Macro image of a tissue slide
c) Patches sampled from the slide
b) Full size slide image

Figure 5.2: An example WSI, the scan includes a macro image that shows the physical microscope slide. The tissue area is selected for scanning and read in a very high resolution. The image size depends on the total tissue area, the average size is about 1.1 GiB.

Pathologists use a combination of these stains to diagnose patients. The dataset we use was created for patient diagnosis. For this process the H&E and CD20 stains were used for all patients, the other stains are not always requested by the pathologist. This is why we do not have a WSI for each patient and every stain, 30 WSIs are missing.

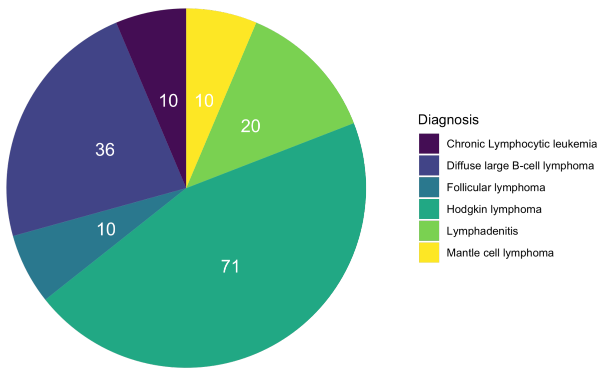


Figure 5.3: Overview of the dataset of lymphoma patients from Kiel

After the first results additional slides were scanned and sent by Sarah Reinke. We received an additional dataset of 30 new patients, each slide scanned twice on two different WSI scanners.

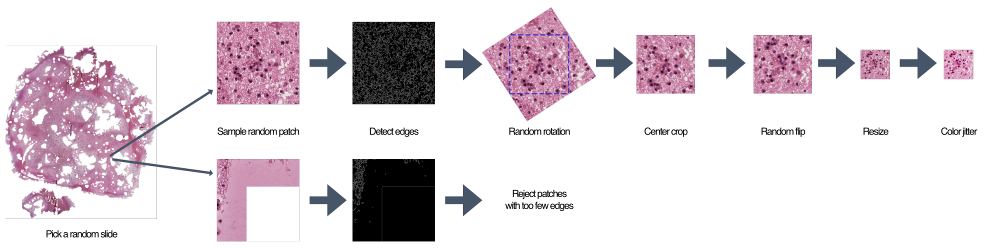


Figure 5.4: Data augmentation pipeline for the neural networks.

5.2.3 Preprocessing and data augmentation

All the slides received were first preprocessed as described in 4.2.5 and 4.2.6. This transforms the proprietary WSI formats into an open SQLite based format and gives us fast random access for reading the slides.

The neural networks we use require a large amount of images for training. The input sizes of the pretrained networks are 244×244 pixels for the ResNet34 [37] or 299×299 for Inception v3 [94]. The slides are orders of magnitudes bigger and the amount of slides we have would not even be close too enough to directly train a neural net. This is why we randomly sample many small squares, we call *patches* from the slides, with which we train the neural networks. The sampling process is shown in Figure 5.4 and looks like this:

1. Pick a fixed sample size, e.g. $200 \times 200\mu m$
2. Convert this size into a pixels count s using the slide metadata. e.g. 880×880 pixels.
3. Read a random region from a WSI with the larger size $\sqrt{2} \times s$
4. Run a canny edge detection [14] on the image, if it contains too few edges we assume we sampled a mostly empty region and resample.
5. Pick a random angle θ and rotate the image around it's center by θ. This is why we initially need to sample a larger region, or else the rotation will crop some pixels.
6. Randomly flip the image vertically of horizontally.
7. Center crop the image to its sample size s, this means removing all pixels not contained in the $s \times s$ square around the center of the image.
8. Resize the image to the size required for the target neural network.
9. Apply a slight color jitter to all the pixels.
10. Normalize the color values for the target network and convert the image to a tensor.

Using this augmentation steps we can easily generate >100 000 unique patches per slide. Furthermore, we ensure that the neural networks do not rely on artifacts from variation of rotation, orientation or stain color. Through the pathology process 4.2.1 the tissue on the slides is already rotated and flipped arbitrarily.

This approach works especially well because the lymph node tissue of lymphoma patients is homogeneous. Given a small section from the lymph node, often done by a needle biopsy, pathologists are still able to diagnose the lymphoma subtype.

5.2.4 Neural networks

We use simple transfer learning on pretrained vision networks on these patches. Multiple different approaches were tried and will be described in 5.2.7. One of the best performing but simple approaches, will be described here in full.

We train a separate neural network for each of the different stains. First a test set of 30 patients was randomly selected, with the other 127 patients as the training set.

A total of $20 \times 4096 = 81,920$ training patches were sampled from the training set using the process described in 5.2.3, this corresponds to 650 patches per slide. Additionally, half as many validation patches were randomly sampled, without any data augmentation. A higher amount of patches did not significantly improve performance. The pretrained neural network *inception v3* [94] was used. It is a convolutional neural network for image classification, pre-trained on the ImageNet dataset [63]. This dataset contains millions of everyday images, each one with a label from a set of 1001 labels, such as “airplane”, “automobile”, “bird”, “cat”, “deer”. This network was adapted by replacing the final linear layer with a new liner layer with just 6 outputs, one for each of the possible diagnoses. The full network can be seen in Figure 5.6.

The network is then trained on all the patches, where each training patch is label with the diagnosis of the slide from which it was sampled. This results in a network trained to classify patches, this means for an input patch it will return a vector of 6 scores, one for each of the diagnoses. This is of course a very simple strategy and is by design inaccurate. Some patches contain no relevant tissue to diagnose and a fixed size for patches is not optimal for all diagnoses. We intentionally ignore these problems because the accuracy of these networks does not need to be perfect. For each patient we have thousands of patches from 5 different stains, as long as most of them are classified correctly the patient will be classified correctly.

5.2.5 Patient diagnosis

After training networks for each stain they are used to diagnose the patients in the test set. Patches are sampled from the slides and fed through the neural networks. We make sure to sample the patches in a way to cover all the tissue of all the slides. The diagnosis results from the networks are tallied up and most common diagnosis for a patient is assigned as the AI diagnosis for the test patient. All the patches essentially vote for the final diagnosis of the patient. In addition, a diagnosis map is created that can be overlaid over the tissue, as seen in Figure 5.5. This show very well which regions of the slides can be diagnosed well and which ones are problematic, pathologists can use their expertise and check the performance of the networks. The full process is summarized in Figure 5.5. In order to create these diagnosis maps for the whole dataset, the AI training and diagnosis process was also run in a 4-fold cross validation, this result in 4 slightly different networks per stain.

5.2.6 Independent test set

In order to check how well these neural networks generalize, a fully independent test set of 30 patients was used, the networks were not re-trained but used for the inference as described in 5.2.5.

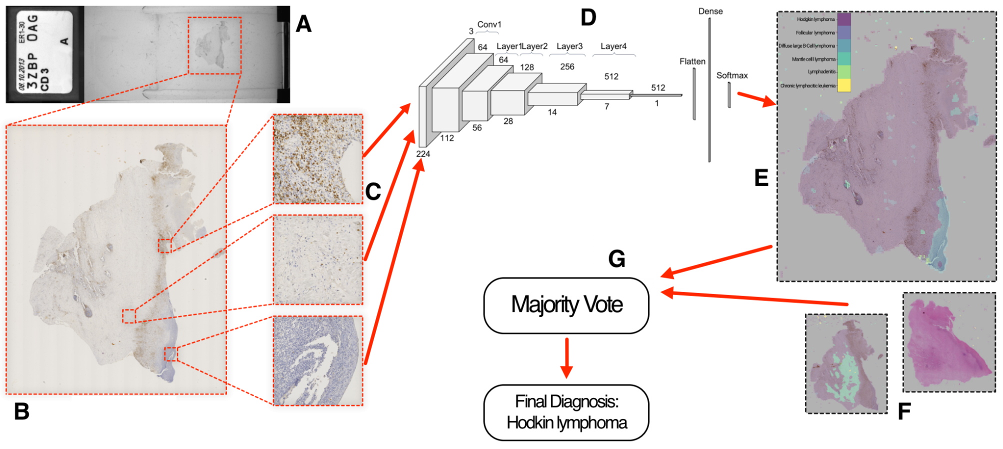


Figure 5.5: AI diagnosis process for an example WSI. The Slide (A) is scanned as a high resolution image (B). From this image patches are sampled (C) and fed through the trained neural networks for classification (D). For each stain a diagnosis map can be created by combining the output of the network from all the patches. (E). For a patient all the available stains are used with a separate network for each (F). All the patches from all the stains are combined into a final diagnosis for the patient (G).

5.2.7 Training variations

The described methodology is the result of an extensive discovery process, on how to best set up the diagnosis architecture. This included the following:

- Replacing the majority voting by a second-level random-forest classifier, trained as part of the cross validation.
- Testing different pre-trained neural networks such as ResNet34 (see Figure 5.6) and ResNet100.

- More aggressive data augmentation and color jitter.
- Sampling more data for training.
- Testing different fixed input sizes, and varying input size as part of data augmentation.
- Allowing for an additional diagnosis label `Unknown`, with a custom error function for backpropagation.
- Including newer data when training the neural networks.
- An independent reproduction of the method by Julian Lorenz as his bachelor thesis [64]
- "Negative" tests, e.g. intentionally training the networks only on bad "background" patches.

5.3 Results

5.3.1 Base dataset

Training on the initial dataset of 157 patients, with a test set of 30 patients resulted in great performance. 60% of patches where classified correctly by the networks. When combining the patch diagnoses into a patient diagnosis all but two patients (93.33%) were classified correctly. These two patients were re-checked by pathologists and are errors by the neural network. When, instead of majority voting, a random forest classifier is used for combining the patch diagnoses, all patients can be classified correctly.

5.3.2 Independent test set

The first independent test set received was from 30 new lymphoma patients, scanned on a different model of scanner. Unfortunately performance was almost completely lost when diagnosing the patients of this test set. Patch level accuracy was down to 20%, barely better than randomly guessing.

The same 30 patients were scanned again with the same scanner as the base data, but this did not significantly improve neural network performance. It resulted in a patch level accuracy of 28%.

Unfortunately none of the changes described in 5.2.7 improved performance significantly. In order to eliminate a metadata or data preparation error the slide labels were validated by pathologists. As the whole workflow and code was independently re-implemented by Julian Lorenz, we also expect this performance is not an error in the code.

Another reason for poor performance are probably technical artifacts in the data. These neural networks strictly optimize diagnosis performance and will use all the information contained in the images. There might be patterns, brightness changes or other artifacts from the preparation or scanning process.

The neural networks might rely on the artifacts for diagnosis, and thus loose performance in an independent test.

The diagnosis maps were valuable for input from pathologists on this topic, see table 5.1 and 5.2 for their conclusions.

Technical Points

- Insufficient tissue detection occurs frequently. Variability in tissue detection between type of stainings (<slide ID>)
- Misclassification occurs in tissue types not trained to AI. e.g. sclerosis and soft tissue as HL (<slide ID>), <slide ID> fatty tissue with blasts as MCL, colonic mucosa as DLBCL <slide ID>
- AI insufficiently recognizes unaffected lymph node areas (partial infiltration) (<slide ID>)
- Unspecific staining is misinterpreted by AI e.g. in necrosis (CD20 <slide ID>)
- AI recognized tissue variably according to staining type. Example necrosis in <slide ID> recognized as tissue in CD20 but not as tissue in HE. Some areas not or only partially stained in CD20 not recognized as tissue (<slide ID>). Defects in tissue not sufficiently recognized or overestimated (both in <slide ID>).
- The overlay of AI results seem to have a tendency of deviating towards outer borders of tissue fragment. <slide ID>.

Table 5.1: Technical points from pathologists Prof. Dr. Wolfram Klapper and Dr. Sarah Reinke, based on the diagnosis maps, slide identifiers redacted

The network's performance was also checked when using an "inverted" patch selection function, where we train the network only on patches we would normally reject as "background" in the pre-selection process. While the networks performed worse, a total loss of performance was not observed. A model interpretability library was used to generate and compare attentions maps on the patch level networks. This did not reveal any obvious artefacts. Fortunately through our diagnosis maps we already generate an "inner", explainable representation of the networks. The comments from pathologists indicate that the networks, are at least somewhat influenced by artefacts in the data.

When mixing in the new test cases with the base dataset, and then creating a new random test set performance also dropped significantly. This indicates that there is variability in the data that the base dataset does not cover.

5.4 Discussion

While the field of AI pathology is huge with hundreds of published models and datasets, AI diagnosis of lymphoma requires somewhat different methodology. For other cancers, such as breast cancers metastasis will form in the lymph nodes and the pathology task is, first of all, detecting the metastasis tissue. The CAMELYON16 challenge on finding breast cancer metastasis [22] alone includes 25 different deep learning based algorithms for this segmentation task. While these approaches informed some methodology with our approach, the key difference for lymphoma is that there are no two clearly distinguished tissue types in the lymph node. In our case the whole lymph node is uniform and the AI task is not image segmentation but image diagnosis. For AI lymphoma cases a preliminary study from Achi. et. al [1] achieved a 95% accuracy with a simpler CNN, for the diagnoses lymphadenitis, DLBCL, Burkitt Lymphoma and CLL. This is similar to the initial performance achieved here. Multiple other findings show that AI models can exhibit a strong decline in performance when evaluated on independent datasets [60] [12].

Miyoshi et. al. [69] improved on this methodology by not randomly sampling tissue patches, but by sampling only from areas annotated by experienced hematopathologists. This is in some ways disadventagious as it requires manual effort from a hematopathologist, but increases robustness of the neutral networks.

5.5 Outlook

While the results from this work did not create a model that generalizes well, the results are nonetheless promising. Based on this work, further research was funded by the Federal Ministry of Education and Research(BMBF) for a federated learning approach to AI pathology [10].

As Miyoshi et. al. [69] show, an obvious weakness of our approach that can be improved is the poor quality of patches we use for training the networks. If we can select better input data, patches that a pathologist would also select for diagnosis, the networks can be improved. Of course this pre-selection should not require manual labor from pathologists but can be treated as another AI task. Preliminary work and data collection on this is already being done, as part of the FDLP project. Visual features can be extracted with powerful new Vision Transformers architectures, such as DINOv2 [76].

A federated approach also allows us to run models on huge datasets, available locally at the pathologies without creating privacy risks for patient data [38].

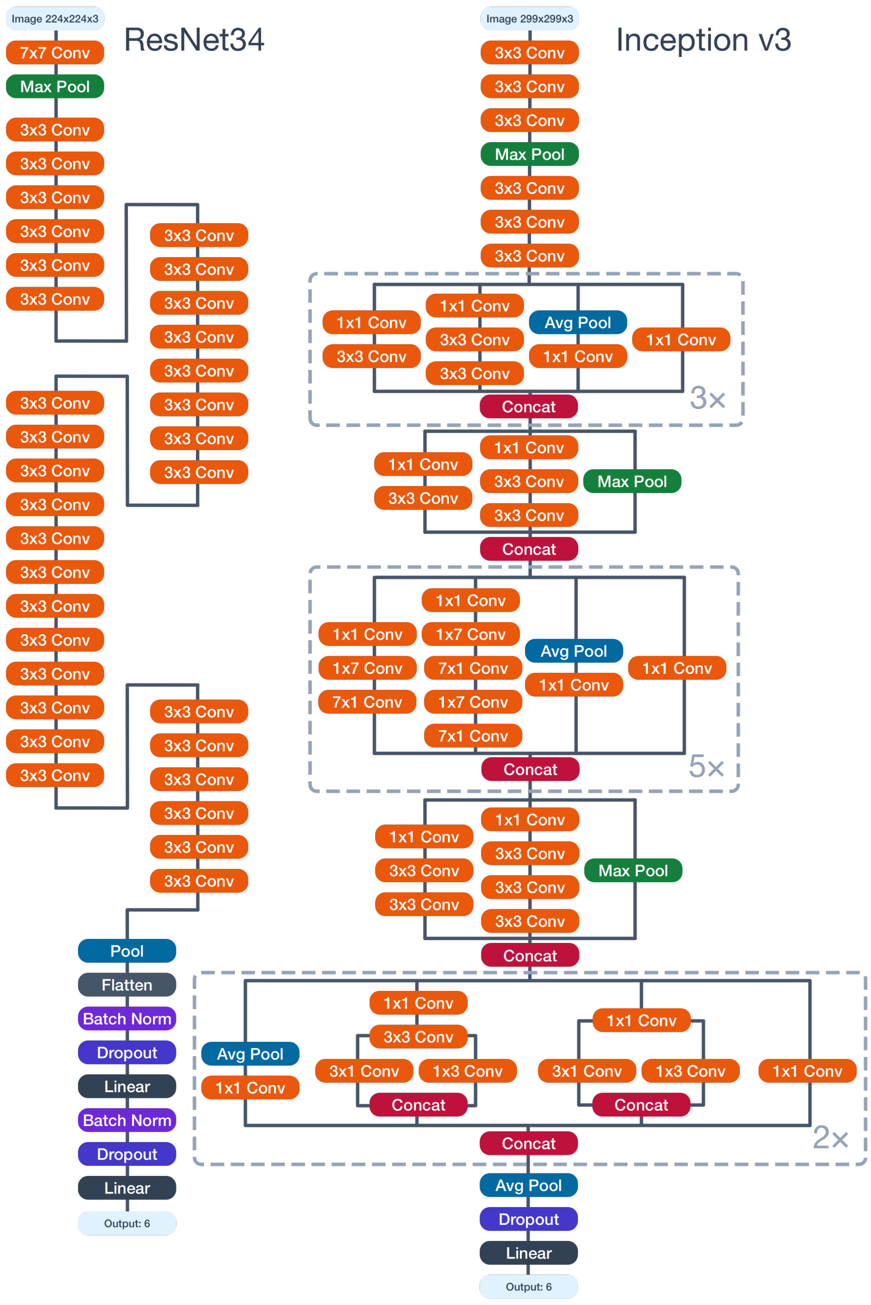


Figure 5.6: Neural networks used for slide diagnosis. *Left:* the structure of the adapted ResNet34 [37] neural network. *Right:* the structure of the Inception v3 network [94]

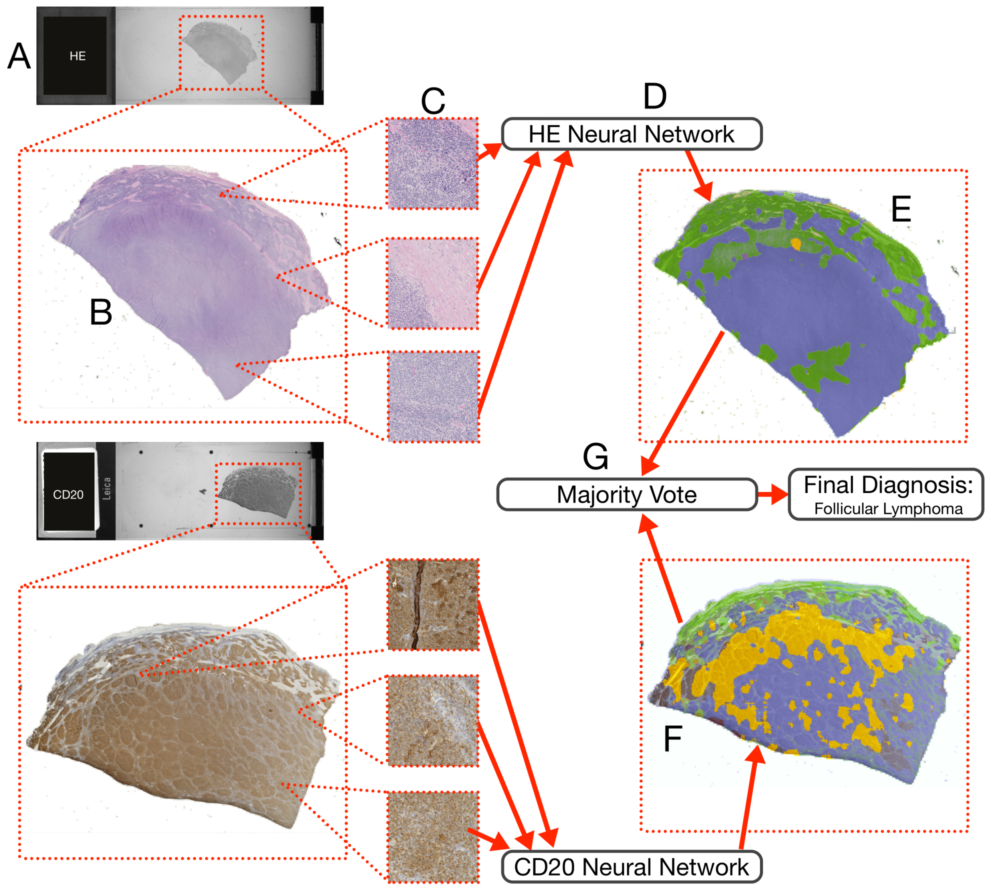


Figure 5.7: The diagnosis pipeline for the base dataset could be reduced to only require two stains (H&E and CD20) with minimal loss of performance. These two stainings are very common for lymph node staining and routinely done by many pathologies.

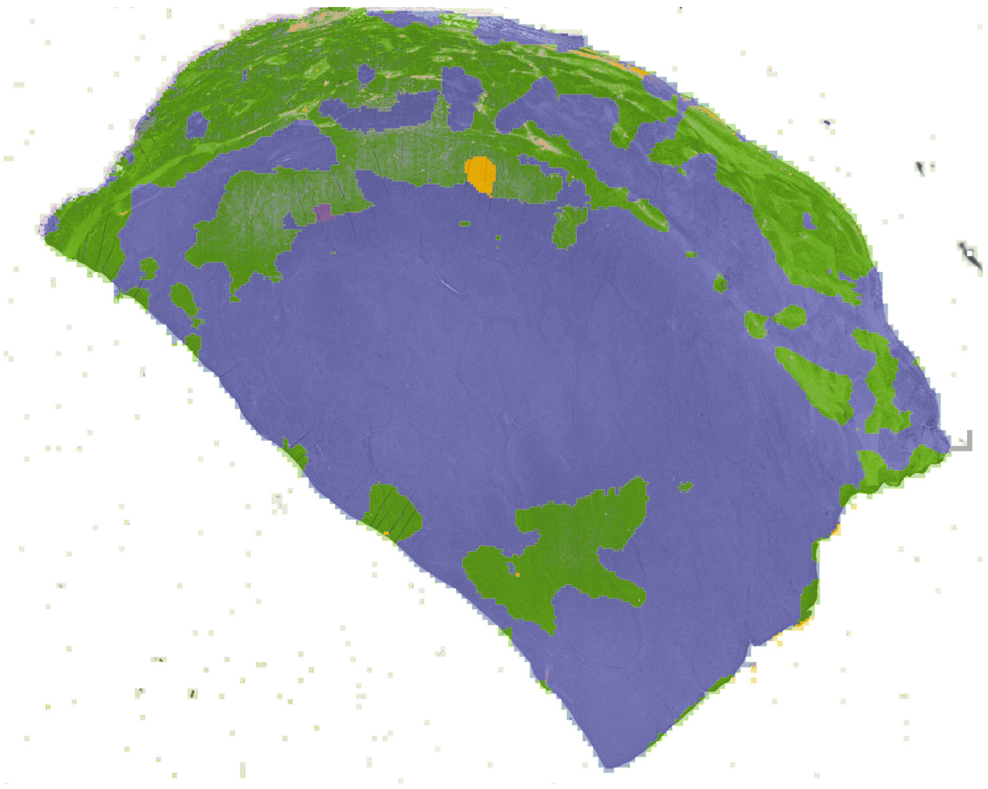


Figure 5.8: The detailed diagnosis map for the H&E stain of Figure 5.7, the maps have the same resolution as the original slide. The overlay can be turned off for checking the tissue. It can be seen that artefacts, such as the small tears in the tissue cause misdiagnoses. The round yellow region is also likely the location where the tissue was picked up with tweezers and damaged.

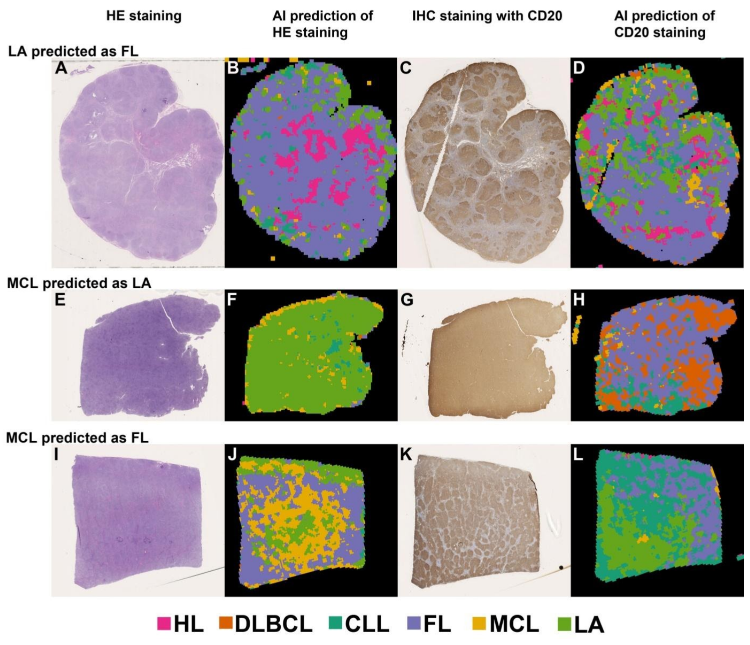


Figure 5.9: Examples of incorrect predictions and their diagnosis maps. This data gives pathologists a good understanding of where the errors happen, and they can look at the slide themselves.

Conclusions

- AI has difficulties in recognition of what seems obvious for a pathologists eye. Most remarkable example is deficiency in recognition of follicles e.g. in <slide ID> a FL almost completely classified as CLL in the CD20 staining despite obvious follicular architecture. In contrary: well recognition of FL diagnosis by AI when follicles are less well defined and almost confluent (<slide ID>, CD20 stain). Artefacts such as labels with a permanent pen (<slide ID> HE stain) or edges of cover slip (<slide ID> CD20) recognized as tissue and classified incorrectly. MCL GC center classified as MCL (<slide ID> CD20)

- AI maps can be visually inspected by pathologist: Hodgkin shows by far the most convincing classification maps when inspected visually (followed by lymphadenitis).

- Obviously immunohistochemical reaction (positive staining) only is a part of the information used by AI: Example for recognition independent of positive staining: Insufficient staining for CD20 in central area correctly recognized as DLBCL (<slide ID>, <slide ID>). Example for incorrect classification due to unexpected negativity for CD20 in DLBCL (<slide ID>).

- Tissue features leading to local misclassification can be partially identified: HL- sclerosis/soft tissue, MCL – foldings, scratches, FL+LyA – T-cell and histiocyte rich but no follicular areas (e.g. in fatty tissue).

- Areas with deviating classification by AI frequently do not show a counterpart by visual inspection. E.g. remnants of GC obvious by visual inspection frequently not recognized by AI as such. Examples: unaffected lymph node areas (<slide ID>), 2 CLL with lots of GC: <slide ID>, <slide ID>.

Table 5.2: Conclusions from pathologists Prof. Dr. Wolfram Klapper and Dr. Sarah Reinke, based on the diagnosis maps, slide identifiers redacted

Bibliography

[1] H. E. Achi, T. Belousova, L. Chen, A. Wahed, I. Wang, Z. Hu, Z. Kanaan, A. Rios, and A. N. D. Nguyen. Automated diagnosis of lymphoma with digital pathology images using deep learning. *Ann Clin Lab Sci*, 49(2): 153–160, Mar 2019.

[2] The Kubernetes Authors. Kubernetes, 2023. URL `https://kubernetes.io`. accessed on March 21, 2024.

[3] The Kubernetes Authors. nfs-subdir-external-provisioner, 2023. URL `https://github.com/kubernetes-sigs/nfs-subdir-external-provisioner`. accessed on March 21, 2024.

[4] The Kubernetes Authors. Kubernetes openid, 2023. URL `https://kubernetes.io/docs/concepts/security/hardening-guide/authentication-mechanisms/#openid-connect-token-authentication`. accessed on March 21, 2024.

[5] Peter Bankhead, Maurice B Loughrey, José A Fernández, Yvonne Dombrowski, Darragh G McArt, Philip D Dunne, Stephen McQuaid, Richard T Gray, and Liam J Murray. Qupath: Open source software for digital pathology image analysis. *Scientific Reports*, 2017. doi: 10.1038/s41598-017-17204-5. URL `https://doi.org/10.1038/s41598-017-17204-5`.

[6] Martin Barisits. Rucio: Scientific data management. *Computing and Software for Big Science*, 2019. doi: 10.1007/s41781-019-0026-3.

[7] Richard Barnes, Jacob Hoffman-Andrews, Daniel McCarney, and James Kasten. Automatic certificate management environment (acme), 2019. URL `https://www.rfc-editor.org/info/rfc8555`. RFC 8555.

[8] Carl Boettiger. An introduction to docker for reproducible research. *SIGOPS Oper. Syst. Rev.*, 49(1):71–79, jan 2015. ISSN 0163-5980. doi: 10.1145/2723872.2723882. URL `https://doi.org/10.1145/2723872.2723882`.

[9] Tim Bray. The JavaScript Object Notation (JSON) Data Interchange Format, 2017. URL `https://www.rfc-editor.org/info/rfc8259`. RFC 8259.

[10] Bundesminiserium für Bildung und Forschung. Fdlp – föderiertes lernen in der lymphompathologie: Infrastruktur, modelle, erweiterungsalgorithmen, detektion von hochrisikopatienten, 2022.

URL https://www.gesundheitsforschung-bmbf.de/de/fdlp-foderiertes-lernen-in-der-lymphompathologie-infrastruktur-modelle-15913.php.

[11] GitLab B.V. Gitlab, 2024. URL https://gitlab.org. accessed on March 21, 2024.

[12] Gabriele Campanella, Matthew G. Hanna, Luke Geneslaw, Allen Miraflor, Vanessa Werneck Krauss Silva, Klaus J. Busam, Edi Brogi, Victor E. Reuter, David S. Klimstra, and Thomas J. Fuchs. Clinical-grade computational pathology using weakly supervised deep learning on whole slide images. *Nat Med*, 25(8):1301–1309, Aug 2019. doi: 10.1038/s41591-019-0508-1.

[13] Brian Campbell, Chuck Mortimore, and Michael B. Jones. Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants, 2015. URL https://www.rfc-editor.org/info/rfc7522. RFC7522.

[14] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8:679–698, 12 1986. doi: 10.1109/TPAMI.1986.4767851.

[15] Canonical Ltd. Ubuntu server, 2023. URL https://ubuntu.com/download/server.

[16] CERN and contributors. Invenio rdm, 2023. URL https://inveniosoftware.org/products/rdm/. accessed on March 21, 2024.

[17] Cloud Native Computing Foundation (CNCF). Container device interface specification, 2023. URL https://github.com/cncf-tags/container-device-interface/blob/main/SPEC.md#version.

[18] Sequelize Contributors. Sequelize, 2023. URL https://sequelize.org. accessed on March 21, 2024.

[19] Tauri Contributors. tauri, 2023. URL https://tauri.app. accessed on March 21, 2024.

[20] Miao Cui and David Y. Zhang. Artificial intelligence and computational pathology. *Laboratory Investigation*, 101(4):412–422, April 1 2021. ISSN 1530-0307. doi: 10.1038/s41374-020-00514-0. URL https://doi.org/10.1038/s41374-020-00514-0.

[21] Stephen Dolan. jq, 2012. URL https://github.com/jqlang/jq. accessed on March 21, 2024.

[22] Babak Ehteshami Bejnordi, Mitko Veta, Paul Johannes van Diest, Bram van Ginneken, Nico Karssemeijer, Geert Litjens, Jeroen A. W. M. van der Laak, and the CAMELYON16 Consortium. Diagnostic Assessment of Deep Learning Algorithms for Detection of Lymph Node

Metastases in Women With Breast Cancer. *JAMA*, 318(22):2199–2210, 12 2017. ISSN 0098-7484. doi: 10.1001/jama.2017.14585. URL `https://doi.org/10.1001/jama.2017.14585`.

[23] Allen Roginsky (NIST) Elaine Barker (NIST). Transitioning the use of cryptographic algorithms and key lengths. Technical Report 800-131a, National Institute of Standards and Technology, Gaithersburg, MD, March 2019.

[24] EMBL-EBI. Ega schemas, 2023. URL `https://ega-archive.org/submission/metadata/ega-schema/`. accessed on March 21, 2024.

[25] etcd Authors. etcd, 2024. URL `https://etcd.io`. accessed on March 21, 2024.

[26] European Parliament and the Council of the European Union. Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679`, 2016. accessed on March 21, 2024.

[27] Roy T. Fielding, Mark Nottingham, and Julian Reschke. Http semantics, 2022. URL `https://www.rfc-editor.org/info/rfc9110`. RFC 9110.

[28] Hal Finney, Lutz Donnerhacke, Jon Callas, Rodney L. Thayer, and Daphne Shaw. OpenPGP Message Format. RFC 4880, November 2007. URL `https://www.rfc-editor.org/info/rfc4880`.

[29] Federal Office for Information Security. Cryptographic mechanisms: Recommendations and key lengths. Technical Report BSI TR-02102-1, Federal Office for Information Security, 2023.

[30] OpenID Foundation. Openid connect core 1.0 incorporating errata set 2, 2023. URL `https://openid.net/specs/openid-connect-core-1_0.html`. accessed on March 21, 2024.

[31] Nextcloud GmbH. Nextcloud, 2023. URL `https://nextcloud.com`. accessed on March 21, 2024.

[32] Adam Goode, Bejamin Gilbert, Jan Harkes, Drazen Jukic, and Mahadev Satyanarayanan. Openslide: A vendor-neutral software foundation for digital pathology. *Journal of Pathology Informatics*, 4(1):27, 2013. ISSN 2153-3539. doi: https://doi.org/10.4103/2153-3539.119005. URL `https://www.sciencedirect.com/science/article/pii/S2153353922006484`.

[33] W3C Group. Web cryptography api, 2023. URL `https://www.w3.org/TR/WebCryptoAPI/`. accessed on March 21, 2024.

[34] Internet Security Research Group(ISRG). Let's encrypt, 2024. URL https://letsencrypt.org. accessed on March 21, 2024.

[35] Hamamatsu Photonics K.K. Ndp.view2 image viewing software, 2018. URL https://www.hamamatsu.com/us/en/product/life-science-and-medical-systems/digital-slide-scanner/U12388-01.html.

[36] Junio Hamano, Shawn Pearce, and Linus Torvalds. Git, 2005. URL https://git-scm.com/. accessed on March 21, 2024.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

[38] Petr Holub, Henning Müller, Michal Bíl, et al. Privacy risks of whole-slide image sharing in digital pathology. *Nature Communications*, 14:2577, 2023. doi: 10.1038/s41467-023-37991-y. URL https://doi.org/10.1038/s41467-023-37991-y.

[39] A. Hosna, E. Merry, J. Gyalmo, et al. Transfer learning: A friendly introduction. *Journal of Big Data*, 9:102, 2022. doi: 10.1186/s40537-022-00652-w. URL https://doi.org/10.1186/s40537-022-00652-w.

[40] Michael Huttner. Dabih source code, 2023. URL https://github.com/spang-lab/dabih. accessed on March 21, 2024.

[41] Michael Huttner. Dabih container, 2023. URL https://hub.docker.com/r/thespanglab/dabih. accessed on March 21, 2024.

[42] Michael Huttner. Pamly source code, 2023. URL https://github.com/spang-lab/pamly-lib.

[43] Michael Huttner. pamly python package, 2023. URL https://pypi.org/project/pamly/.

[44] Michael Huttner. pamly rust crate, 2023. URL https://lib.rs/crates/pamly.

[45] Michael Huttner and Tobias Schmidt. Uni regensburg notebook converter, 2023. URL https://github.com/spang-lab/urnc. accessed on March 21, 2024.

[46] Michael Huttner, Claudio Lottaz, Christian Kohler, and Rainer Spang. Lyra – containerized microservices for browsing shared biomedical data, 2020. URL https://arxiv.org/abs/2005.12905.

[47] Michael Huttner, Jakob Simeth, Renato Liguori, Fulvia Ferrazzi, and Rainer Spang. dabih - encrypted data storage and sharing platform, 2024.

[48] Docker Inc. Spang lab dockerhub, 2023. URL `https://hub.docker.com/u/thespanglab`. accessed on March 21, 2024.

[49] Docker Inc. Docker, 2024. URL `https://www.docker.com`. accessed on March 21, 2024.

[50] GitHub Inc. Github, 2024. URL `https://github.com`. accessed on March 21, 2024.

[51] The Open Container Initiative. Open container initiative image specification, 2024. URL `https://github.com/opencontainers/image-spec`. accessed on March 21, 2024.

[52] The Open Container Initiative. Open container initiative runtime specification, 2024. URL `https://github.com/opencontainers/runtime-spec`. accessed on March 21, 2024.

[53] International Organization for Standardization. Iso/iec 18004:2015 qr code bar code symbology specification, 2015.

[54] Hidetake Iwata. kubelogin, 2022. URL `https://github.com/int128/kubelogin`.

[55] Michael Huttner Jakob Simeth. Datatomb, 2021. URL `https://github.com/spang-lab/datatomb`. accessed on March 21, 2024.

[56] Michael Huttner Jakob Simeth. glacier datatomb cli, 2021. URL `https://gitlab.spang-lab.de/jsimeth/glacier`. accessed on March 21, 2024.

[57] Michael B. Jones. JSON Web Key (JWK). RFC 7517, May 2015. URL `https://www.rfc-editor.org/info/rfc7517`.

[58] JS Foundation. webpack, 2023. URL `https://webpack.js.org`. accessed on March 21, 2024.

[59] Burt Kaliski. Public-Key Cryptography Standards (PKCS) #8: Private-Key Information Syntax Specification Version 1.2. RFC 5208, May 2008. URL `https://www.rfc-editor.org/info/rfc5208`.

[60] Inho Kim, Kyunghee Kang, Yosep Song, and Tae Joo Kim. Application of artificial intelligence in pathology: Trends and challenges. *Diagnostics (Basel)*, 12(11):2794, Nov 2022. doi: 10.3390/diagnostics12112794.

[61] Philip Koopman. 32-bit cyclic redundancy codes for internet applications. In *Proceedings International Conference on Dependable Systems and Networks*, pages 459–468, 2002. ISBN 978-0-7695-1597-7. doi: 10.1109/DSN.2002.1028931. URL `https://doi.org/10.1109/DSN.2002.1028931`. Verification of Castagnoli's results by exhaustive search and some new good polynomials.

[62] Tailwind Labs. tailwindcss, 2023. URL `https://tailwindcss.com`. accessed on March 21, 2024.

[63] Kai Li Li Fei-Fei, Jia Deng. Imagenet: Constructing a large-scale image database. *Journal of vision*, 9, 8 2009. doi: 10.1167/9.8.1037.

[64] Julian Lorenz. Bachelor thesis: Cancer diagnosis on lymph node images using deep learning, 2019.

[65] Seafile Ltd. Seafile, 2023. URL `https://www.seafile.com/en/home/`. accessed on March 21, 2024.

[66] Fergal J Martin, M Ridwan Amode, Alisha Aneja, Olanrewaju Austine-Orimoloye, Andrey G Azov, Barnes, et al. Ensembl 2023. *Nucleic Acids Research*, 51(D1):D933–D941, 11 2022. ISSN 0305-1048. doi: 10.1093/nar/gkac958. URL `https://doi.org/10.1093/nar/gkac958`.

[67] Nicholas D Matsakis and Felix S Klock II. The rust language. In *ACM SIGAda Ada Letters*, volume 34, pages 103–104. ACM, 2014.

[68] Meta. React, 2023. URL `https://react.dev`. accessed on March 21, 2024.

[69] H. Miyoshi, K. Sato, Y. Kabeya, S. Yonezawa, H. Nakano, Y. Takeuchi, I. Ozawa, S. Higo, E. Yanagida, K. Yamada, K. Kohno, T. Furuta, H. Muta, M. Takeuchi, Y. Sasaki, T. Yoshimura, K. Matsuda, R. Muto, M. Moritsubo, K. Inoue, T. Suzuki, H. Sekinaga, and K. Ohshima. Deep learning shows the capability of high-level computer-aided diagnosis in malignant lymphoma. *Lab Invest*, 100(10):1300–1310, Oct 2020. doi: 10.1038/s41374-020-0442-3.

[70] National Institute of Standards and Technology. Recommendation for block cipher modes of operation: Methods and techniques. Technical Report 800-38A, National Institute of Standards and Technology, Gaithersburg, MD, 2001. NIST Special Publication 800-38A.

[71] National Institute of Standards and Technology. Secure hash standard (shs). Technical Report FIPS PUB 180-4, National Institute of Standards and Technology, Gaithersburg, MD, 2015. FIPS 180-4.

[72] Bastiaan G. L. Nelissen, Joost A. van Herwaarden, Frans L. Moll, Paul J. van Diest, and Gerard Pasterkamp. Slidetoolkit: An assistive toolset for the histological quantification of whole slide images. *PLoS ONE*, 2014. doi: 10.1371/journal.pone.0110289. URL `https://doi.org/10.1371/journal.pone.0110289`.

[73] NVIDIA Corporation. Nvidia container toolkit, 2021. URL `https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/latest/index.html`.

[74] University of North Carolina at Chapel Hill and Dell EMC. Metalnx, 2015. URL https://github.com/irods-contrib/metalnx-web. accessed on March 21, 2024.

[75] Open Source Initiative. The open source definition, 2006. URL http://opensource.org/docs/osd. accessed on March 21, 2024.

[76] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2024.

[77] Podman. Podman, 2024. URL https://podman.io. accessed on March 21, 2024.

[78] J. Postel. Internet protocol, 1981. URL https://www.rfc-editor.org/info/rfc791. RFC 791.

[79] Preston-Werner. Semantic versioning 2.0.0, 2013. URL https://semver.org. accessed on March 21, 2024.

[80] Project Jupyter Contributors. Zero to jupyterhub with kubernetes, 2023. URL https://github.com/jupyterhub/zero-to-jupyterhub-k8s. accessed on March 21, 2024.

[81] PyO3 Contributors. Pyo3: Rust bindings for python, 2023. URL https://github.com/PyO3/pyo3.

[82] Arcot Rajasekar, Terrell Russell, Jason Coposky, Antoine de Torcy, Hao Xu, Michael Wan, Reagan W. Moore, Wayne Schroeder, Sheau-Yen Chen, Mike Conway, and Jewel H. Ward. *The integrated Rule-Oriented Data System (iRODS 4.0) Microservice Workbook*. CreateSpace Independent Publishing Platform, North Charleston, SC, USA, 1st edition, 2015. ISBN 1511732776. doi: 10.5555/2834461.

[83] RSA Laboratories. Public-key cryptography standards (pkcs) #1: Rsa cryptography specifications version 2.1. Technical report, RSA Laboratories, 2003. URL https://datatracker.ietf.org/doc/html/rfc3447. RFC 3447.

[84] J.H. Saltzer and M.D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975. doi: 10.1109/PROC.1975.9939.

[85] F. Sanger, A.R. Coulson, B.G. Barrell, A.J.H. Smith, and B.A. Roe. Cloning in single-stranded bacteriophage as an aid to rapid dna sequencing. *Journal of Molecular Biology*, 143(2):161–178, 1980. ISSN 0022-2836. doi: https://doi.org/10.1016/0022-2836(80)90196-5. URL https://www.sciencedirect.com/science/article/pii/0022283680901965.

[86] Eric W Sayers, Evan E Bolton, J Rodney Brister, Kathi Canese, Jeonghoon Chan, Donald C Comeau, Richard Connor, Karen Funk, Christiam Kelly, Seungjin Kim, Tom Madej, Aron Marchler-Bauer, Christopher Lanczycki, Stephen Lathrop, Zhiyong Lu, Francoise Thibaud-Nissen, Terence Murphy, Lon Phan, Yevgeniya Skripchenko, Tony Tse, Jane Wang, Robin Williams, Belem Trawick, Kim D Pruitt, and Stephen T Sherry. Database resources of the national center for biotechnology information. *Nucleic Acids Research*, 50(D1):D20–D26, 2022. doi: 10.1093/nar/gkab1112.

[87] Juergen Schmidhuber. Annotated history of modern ai and deep learning. *arXiv*, 12 2022. doi: 10.48550/arXiv.2212.11279.

[88] Marian Schön. Celloscope, 2021. URL `https://github.com/MarianSchoen/celloscope`.

[89] Marian Schön, Jakob Simeth, Paul Heinrich, Franziska Görtler, Stefan Solbrig, Tilo Wettig, Peter J. Oefner, Michael Altenbuchinger, and Rainer Spang. Dtd: An R package for digital tissue deconvolution. *Journal of Computational Biology*, pages 386–389, March 2020. doi: 10.1089/cmb.2019.0469. URL `http://doi.org/10.1089/cmb.2019.0469`.

[90] Alexander Senf, Richard Davies, Fabrice Haziza, John Marshall, Juan Troncoso-Pastoriza, Oliver Hofmann, and Thomas M. Keane. Crypt4gh: A file format standard enabling native access to encrypted data. *Bioinformatics (Oxford, England)*, 37(17):2753–2754, 2021. doi: 10.1093/bioinformatics/btab087.

[91] Jakob Simeth. digger: Datatomb r package, 2021. URL `https://github.com/spang-lab/DiggeR`. accessed on March 21, 2024.

[92] Rainer Spang, Claudio Lottaz, and Michael Huttner. Genomic data analysis, 2024. URL `https://www.spang-lab.de/students/data-analysis`. accessed on March 21, 2024.

[93] SQLite Consortium. Sqlite, 2023. URL `https://sqlite.org`. accessed on March 21, 2024.

[94] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016. doi: 10.1109/CVPR.2016.308.

[95] The FIDS Phd Students. Developers skills, 2024. URL `https://fids.git-pages.uni-regensburg.de/developer-skills/00_developer_skills.html`. accessed on March 21, 2024.

[96] The German Human Genome-Phenome Archive. German human genome-phenome archive, 2022. URL `https://github.com/ghga-de/ghga-metadata-schema/releases/tag/0.7.0`. accessed on March 21, 2024.

[97] The OpenBSD Project. Openssh, 2023. URL https://github.com/openssh/openssh-portable.

[98] The OpenJS Foundation. Node.js, 2023. URL https://nodejs.org. accessed on March 21, 2024.

[99] The PostgreSQL Development Group. Postgresql, 2023. URL https://postgresql.org. accessed on March 21, 2024.

[100] The R Foundation. The r project for statistical computing, 2024. URL https://www.r-project.org. accessed on March 21, 2024.

[101] The TRR305 Researchers. Striking a moving target: From mechanisms of metastatic organ colonisation to novel systemic therapies, 2022. URL https://www.trr305.de/research-program. accessed on March 21, 2024.

[102] UC Berkeley Data Science Division. Data 8: The foundations of data science, 2023. URL http://www.data8.org. accessed on March 21, 2024.

[103] Vercel. next.js, 2023. URL https://nextjs.org. accessed on March 21, 2024.

[104] Mark Wilkinson, Michel Dumontier, IJsbrand Aalbersberg, et al. The FAIR guiding principles for scientific data management and stewardship. *Scientific Data*, 3:160018, 2016. doi: 10.1038/sdata.2016.18. URL https://doi.org/10.1038/sdata.2016.18.

[105] Mark Wilkinson, Michel Dumontier, IJsbrand Aalbersberg, et al. The fair principles, 2024. URL https://www.go-fair.org/fair-principles/. accessed on March 21, 2024.

[106] Kurt Zeilenga. Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map, 2006. URL https://www.rfc-editor.org/info/rfc4510. RFC 4510.

List of Figures

List of Tables