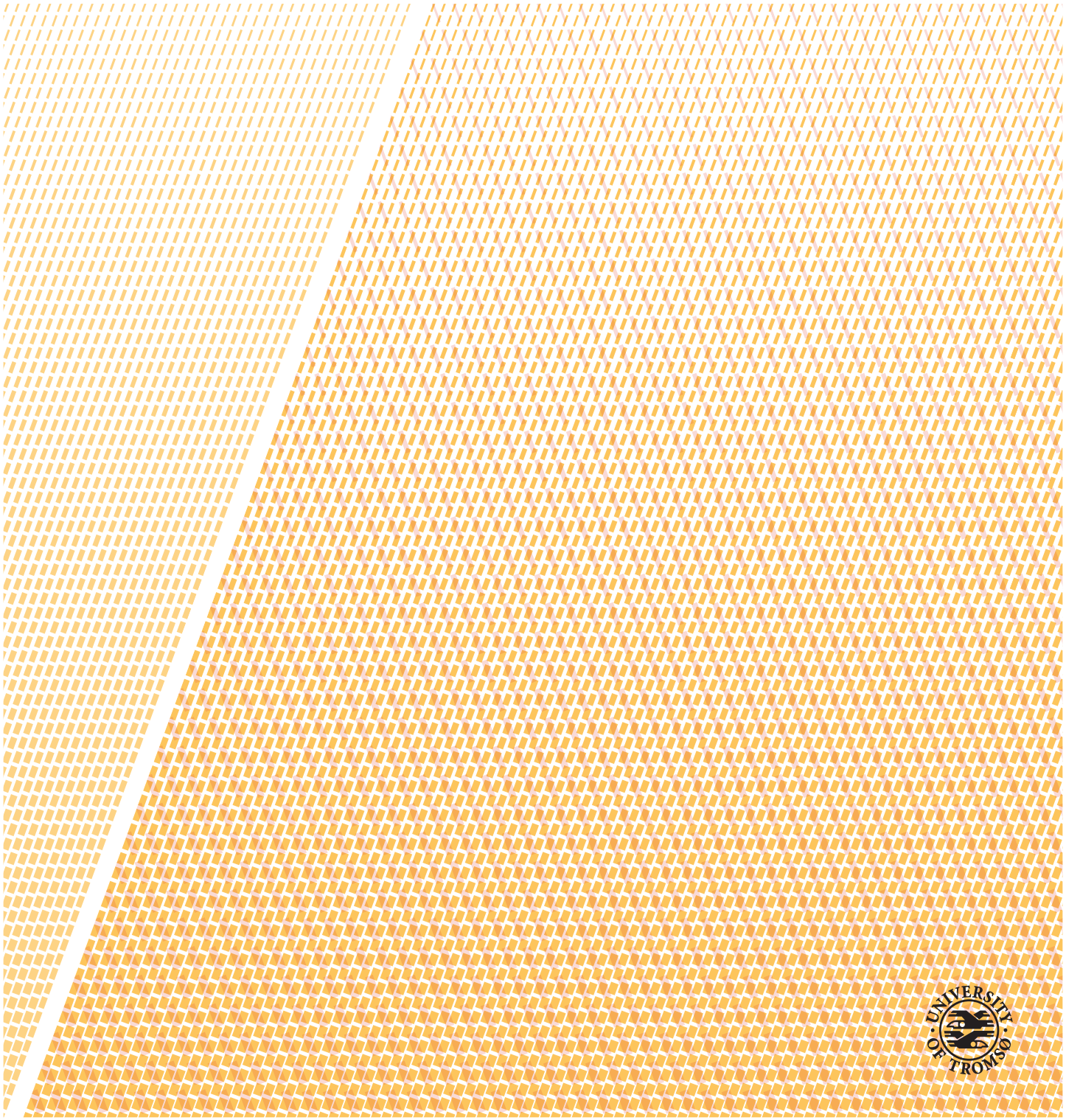
UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

Faculty of Science and Technology
Department of Computer Science

Reproducible Analysis and Exploration of High-Throughput Biological Datasets

Bjørn Fjukstad

A dissertation for the degree of Philosophiae Doctor



“Ta aldri problemene på forskudd, for da får du dem to ganger, men ta gjerne seieren på forskudd, for hvis ikke er det alt for sjelden du får oppleve den.”

–Ivar Tollefsen

Abstract

There is a rapid growth in the number of available biological datasets due to the advent of high-throughput data collection instruments combined with cheap compute infrastructure. Datasets from modern instruments enable analysis of biological data at different levels, from small DNA sequences through larger cell structures, and up to the function of entire organs. These new datasets have brought the development of new software tools and packages for analysis. This leads to the potential for novel insights to the underlying biological mechanisms in the development and progression of diseases such as cancer.

The heterogeneity of these biological datasets require researchers to tailor the exploration and analyses using a range of different tools and systems. However, despite the need to use a range of tools, few of these provide standard interfaces for analyses implemented using different programming languages and frameworks. In addition, because of the many tools, input parameters, and reference to databases, it is difficult to correctly report the correct details of an analysis. The lack of such details complicates reproducing original results and reusing the analyses on new datasets. This increases both analysis time and leaves unrealized potential for scientific insights.

This dissertation argues that we can develop unified systems for reproducible exploration and analysis high-throughput biological datasets. We propose an approach, SME, that orchestrates the execution of analysis pipelines and data exploration applications. We realize SMEs using software container technologies together with well-defined interfaces, configuration, and orchestration. It simplifies the development of such applications, and provides detailed information to reproduce the analyses.

Through our approach we have developed different applications for analyzing high-throughput DNA sequencing datasets, and exploring gene expression data integrated with questionnaire, registry, and online databases. Our evaluation shows how we effectively capture provenance in analysis pipelines and exploration applications. This simplifies reproducing and sharing of methods and tools.

Acknowledgements

Insert acks here.

Contents

List of Figures

List of Tables

List of Abbreviations

CLI Command-line Interface

CSV Comma-separated values

CWL Common Workflow Language

DAG directed acyclic graph

DNA Deoxyribonucleic acid

GATK Genome Analysis Toolkit

GB Gigabyte

GUI Graphical User Interface

HTS High-throughput Sequencing

JSON JavaScript Object Notation

KEGG Kyoto Encyclopedia of Genes and Genomes

MIxT Matched Interactions Across Tissues

NGS Next-generation Sequencing

NOWAC Norwegian Women and Cancer

PFS Pachyderm File System

PPS Pachyderm Processing System

RNA Ribonucleic acid

SCM source code management

SME Small Modular Entity

SNP Single Nucleotide Polymorphism

SR Systemic Response

WES Whole-exome sequencing

WGS Whole-genome sequencing

XML Extensible Markup Language

YAML YAML Ain't Markup Language

1

Introduction

There is a rapid growth in the number of available biological datasets due to the decreaseing cost of data collection. This brings opportunities to gain novel insights to the underlying biological mechanisms in the development and progression of diseases such as cancer, possibly leading to the development of novel diagnostic tests or drugs for treatment. The wide range of different biological datasets has led to the development of hundreds of software packages and systems to explore and analyze these datasets. However, there are few systems that are designed with the full analysis process in mind, from raw data into interpretable and reproducible results. While existing systems are used to provide novel insights in diseases, there is little emphasis on reporting and sharing information about its tools, their versions, input parameters, and other information that can help others use the same known methods on their own datasets. This leads to unnecessary difficulties to reuse known methods, and reproducing the analyses, which leads to a longer analysis process and therefore unrealized potential for scientific insights.

There are four main challenges for researchers to analyze and explore biological datasets. These challenges are common for large datasets such as high-throughput sequencing data that require long-running, deep analysis pipelines, as well as smaller datasets, such as microarray data, that require complex, but short-running analysis pipelines. The first is the time and knowledge required to find and set up the necessary analysis tools to start analyzing a modern biological dataset. The second is recording full provenance in the data exploration process. This includes ensuring the correct input parameters, tool

versions, database versions, and dataset versions when analyzing, and reporting analysis results to enable reproducible science. A third challenge is efficiently exploring the results of an analysis interactively. This includes developing tools that can efficiently visualize the heterogeneous resulting datsets and integrate them with known biology from online databases. The final challenge is reusing the analysis pipelines and exploration tools on new datasets and research questions.

As a result, there are a wealth of specialized approaches and systems to analyze modern biological data. Systems such as Galaxy[1] provide simple Graphical User Interfaces (GUIs) to set up and run analysis pipelines. However it is difficult to install and maintain, and less flexible for explorative analyses where it is necessary to try out new tools and different tool configurations.[2] With R and its popular package repository Bioconductor, researchers can select from a wide range of packages to tailor their analyses. These provide flexible environments but makes it necessary for the analyst to manually record information about data, tools, and tool versions. Systems such as Pachyderm[1] or the Common Workflow Language (CWL)[3] and its different implementations, can help users with standardizing the description and sharing of analysis pipelines. However, many of these require complex compute infrastructure and are too cumbersome to set up. Shiny and OpenCPU provide frameworks for application developers to build systems to interactively explore results from statistical analyses. These are useful to build exploration applications that integrate with statistical analyses. With the addition of new datasets and methods every year, it seems that analysis of biological data requires a wide array of different tools and systems.

This dissertation argues that, instead, we can facilitate development of reproducible analyses and data exploration systems of high-throughput biological data thorugh the integration of disperate systems and data. In particular, we show how software container technologies together with well-defined interfaces, configurations, and orchestration provide the necessary foundation for these systems. This allows for easy development and sharing of specialized analysis systems.

The resulting approach, which we have called SME, has several key advantages when implementing systems to analyze and explore biological data:

- It enables reproducible research by packaging applications and tools within containerized environments. This enables sharing of tools and ensures correct versions.

- It enables applications to use tools written in any programming language,

1. `pachyderm.io`

using open standards to communicate between tools and systems.

- Through software container technology it becomes a simple task to deploy and scale up such applications.

- It simplifies the sharing of analysis pipelines and workflows across different research teams and systems. This reduces the time-to-interpretation for biological datasets.

In collaboration with researchers in systems epidemiology and precision medicine we developed a set of applications and systems necessary to organize, analyze, and interpret their datasets. From these systems we extraplolated a set of general design principles to form a unified approach. We evaluate this approach through these systems using real datasets to show its viability.

From a longer-term perspective we discuss the general patterns for implementing reproducible data analysis systems for use in biomedical research. As more datasets are produced every year, research will depend on the simplicity of the systems for analyzing these, and that they provide the necessary functionality to reproduce and share the analysis pipelines.

Thesis statement: A unified development model based on software container infrastructure can efficiently provide reproducible and easy to use environments to develop applications for exploring and analyzing biological datasets.

1.1 Problems with Data Analysis and Exploration in Bioinformatics

Today shell scripts are still the de facto standard building analysis pipelines in bioinformatics. This comes from the familiarity of the shell environment and the Command-line Interface (CLI) of the different tools. However, is a move towards using more sophisticated approaches to analyze biological datasets through workflow and pipeline mangers such as Snakemake[4], and the different implementations of the CWL[**?**] such as Galaxy[1] and Toil[5]. These simplify setting up and executing the analysis pipeline pipeline. However, these tools still have their limitations, such as maintenance and tool updates. For exploring biological data there are a range of tools, such as Cytoscape[**?**] and Circos[**?**], that support importing an already-analyzed dataset to visualize and browse the data. One problem with these is that they are decoupled from the analysis, making it difficult to retrace the underlying analyses.

Although there are efforts to develop tools to help researchers explore and analyze biological datasets, they current tools have several drawbacks:

1. **Standardization:** Because of the specialized nature of each data analysis tool, a complete system for exploring or analyze biological data will have to combine these. The tools provide different interfaces and processing data using a combination of these often require data wrangling.

2. **Decoupling:** Data exploration tools are often decoupled from the statistical analyses. This often makes it a difficult to document and retrace the analyses behind the results.

3. **Complexity:** Analyses that start as a simple script quickly become more complex to maintain and develop as developers add new functionality to the analyses.

4. **Reusability:** Data exploration tools are often developed as a single specialized application, making it difficult to reuse parts of the application for other analyses or datasets. This leads to duplicate development effort and abandoned projects.

5. **Reproducibility:** While there are tools for analyzing most data types today, these require the analyst to manually record versions, input parameters, and reference databases. This makes analysis results difficult to reproduce because of the large number of variables that may impact the results.

Because of these drawbacks, a unified approach for reproducible data analysis and exploration would have significant benefits to reduce the time-to-interpretation of biological datasets.

1.2 Small Modular Entities (SMEs)

In collaboration with researchers in systems epidemiology and biology we have developed an approach for designing applications for three specific use cases. The first is to manage and standardize the analysis of datasets from a large population-based cohort, NOWAC. The second is to enable interactive exploration of these datasets. The final use case is to develop pipelines for analyzing sequencing datasets for use in a precision medicine setting. Although these use cases require widely different systems with different requirements, the applications share common design patterns. Figure 1.1 shows the applications we have developed and the underlying systems.

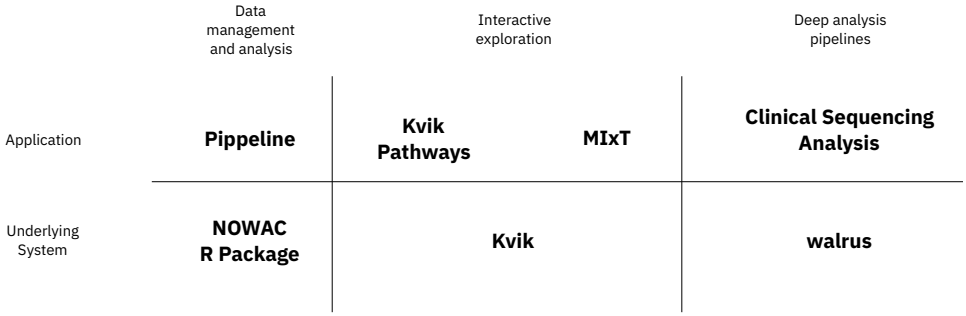


Figure 1.1: The applications and their underlying systems discussed in this thesis.

We discuss how this approach suites applications in the different use cases separately before highlighting the similarities.

1.2.1 Data management and analysis

Modern epidemiological studies integrate traditional questionnaire data with information from public registries biological datasets. These often span multiple biological levels, i.e. different data types and collection sites. While traditional questionnaire datasets require few specialized analysis tools because of the relatively simple nature of the data, biological datasets require specialized tools for reading, analyzing, and interpreting the data. Package repositories such as Bioconductor[**?**] provide a wealth of packages for analyzing these datasets. These packages typically provide analysis tools, example data, and comprehensive documentation. While the analysis code can be shared within projects, the datasets are often stored on in-house databases or shared file systems with specialized permissions. Together the packages and datasets form building blocks for researchers to build their analyses on top of. They can compose their analyses using packages that fit their needs. We wanted the analysis code in the NOWAC study to be one such building block. Therefore we combined the datasets from the NOWAC cohort with documentation, analysis scripts, and integration with registry datasets, into a single package. This

approach simplifies the researcher's first steps in the analysis of the different data in our study. On top of the NOWAC package we then implemented a pre-processing pipelining tool named Pippeline.

Inspired by the ecosystem of packages in the R programming language we implemented our approach as the NOWAC R package. Users simply install the package and get access to documentation, datasets, and utility functions for analyzing datasets related to their area of research. We version control both code and the data, making it possible to track changes over time as the research study evolves. Pippeline is a web-based interface for running the standardized preprocessing steps before analyzing gene expression datasets in the NOWAC cohort.

1.2.2 Deep Analysis Pipelines

Analysis of high-throughput sequencing datasets requires deep analysis pipelines with a large number of steps that transform raw data into interpretable results[6]. There are a large number of tools available to perform the different processing steps, written in a wide range of programming languages. The tools, and their dependencies, can be difficult to install, and they require users to correctly manage a range of input parameters that affects the output results. With software container technology it is a simple task for users to share container images with analysis tools pre-installed. Then, by designing a text-based specification for the analyses, we can orchestrate the execution of an entire analysis pipeline and record the flow of data through the pipeline. As with the previous use case an analysis is built by composing smaller entities, or tools, into a complete pipeline.

We implemented the approach in walrus, a tool that lets users create and run analysis pipelines. In addition, it tracks full provenance of the input, intermediate, and output data, as well as tool parameters. With *walrus* we have successfully built analysis pipelines to detect somatic mutations in breast cancer patients, as well as an Ribonucleic acid (RNA)-seq pipeline for comparison with gene expression datasets.

1.2.3 Interactive Data Exploration Applications

The final results from an analysis pipeline require researchers to investigate and evaluate the final output. In addition, it may be useful to explore the analysis parameters and re-run parts of the analyses. As with analysis pipelines, there are complete exploration tools as well as software libraries to develop custom applications for exploration of analysis results. The tools often require

users to import already analyzed datasets but provide interactive visualizations and point-and-click interfaces to explore the data. Users with programming knowledge can use the wealth of software packages for visualization within languages such as R or Python. Frameworks such as BioJS[7] now provide developers with tools to develop web applications for exploring biological datasets. It is apparent these types of systems also consist of multiple smaller components that together can be orchestrated into a single application. These applications typically consist of three major parts: (i) data visualization; (ii) integration with statistical analyses and datasets; and (iii) integration with online databases. While each of these are specialized for each type of data exploration application, they share components that can be reused across different types of applications.

To integrate with statistical analyses and datasets, we wrote an interface to the R programming language, that would allow us to interface with the wealth of existing software packages, e.g. the NOWAC package, for biological data analyses from a point-and-click application. New data exploration applications could access analyses directly through this interface, removing the previous decoupling between the two. We followed the same approach to integrate with online databases. We could standardize the interface from the applications to the different databases, and implement an application on top of these.

We implemented all components as a part of *Kvik*, a collection of packages to develop new data exploration applications. Kvik allows applications written in any modern programming language to interface with the wealth of bioinformatics packages in the R programming language, as well as information available through online databases. To provide reproducible execution environments we packaged these interfaces into software containers that can be easily deployed and shared. We have used Kvik to develop the MIXT system[8] for exploring and comparing transcriptional profiles from blood and tumor samples in breast cancer patients, in addition to applications for exploring biological pathways[9].

1.2.4 Similarity

The above approaches to build systems to analyze and explore biological data, either as interactive applications or batch processing pipelines share the same design principles. In all areas we break down the systems in to small modular entities, e.g. a tool, and package these into software containers which are then orchestrated together. These containers are configured and communicate using open protocols that make it possible to interface with them using any programming language. We can keep track of the configuration of the containers and their orchestration using software versioning systems,

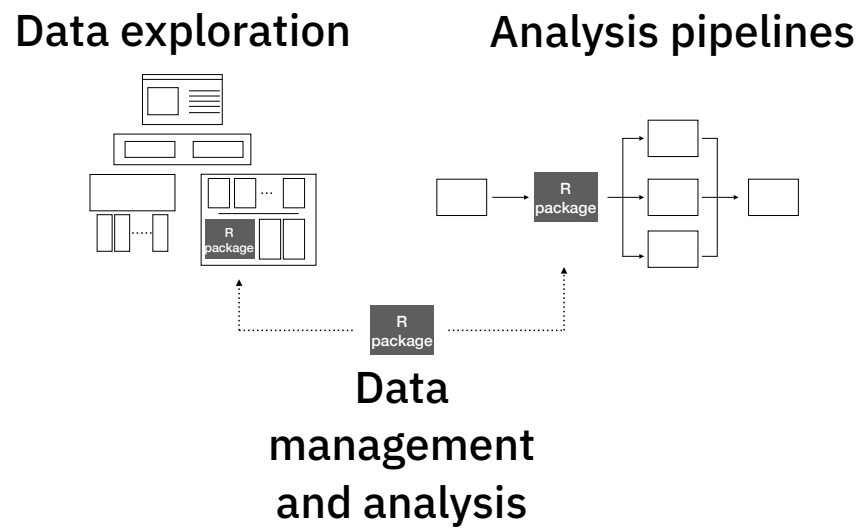


Figure 1.2

and provide the necessary information to reproduce analyses or a complete system.

1.3 Systems Developed with SMEs

In this section we detail the different systems we have built using SMEs, specifically implemented on top of walrus and Kvik.

1.3.1 TODO:NOWAC R PACKAGE PIPELINE

1.3.2 Deep Analysis Pipelines

The first system we built on top of walrus was a pipeline to analyze a patient's primary tumor and adjacent normal tissue, including subsequent metastatic lesions.[?] We packaged the necessary tools for the analyses into software containers and wrote a pipeline description with all the necessary data processing steps. Some of the steps required us to develop specialized scripts to generate customized plots, but these were also wrapped in a container. From the analyses we discovered, among other findings, inherited germline mutations that are

recognized to be among the top 50 mutations associated with an increased risk of familial breast cancer. These were then shared with the treating oncologists to aid the treatment plan.

The second analysis pipeline we implemented was to enable comparison of a RNA-seq dataset to microarray gene expression values collected from the same samples. The pipeline preprocesses the RNA dataset for all samples, and generates transcript quantifications. As with the first pipeline we used existing tools together with specialized analysis scripts packaged into a container to ensure that we could reproduce the execution environments.

1.3.3 Interactive Data Exploration Applications

The first interactive data exploration application we built was Kvik Pathways. It allows users to explore gene expression data from the NOWAC cohort in the context of interactive pathway maps.[9] It is a web application that integrates with the R programming language to provide an interface to the statistical analyses. We used Kvik Pathways to repeat the analyses in a previous published project that compared gene expression in blood from healthy women with high and low plasma ratios of essential fatty acids.[10]

From the first application it became apparent that we could reuse parts of the application in the implementation of later systems. In particular, the interface to run analyses as well as the integration with the online databases could be implemented as services, packaged into containers, and reused in the next application we developed. Both of these were designed and implemented in Kvik, which could then be used and shared later.

The second application we built the MIXT web application. A system to explore and compare transcriptional profiles from blood and tumor samples in breast cancer patients. The application is built to simplify the exploration of results from the Matched Interactions Across Tissues (MIxT) study. Its goal was to identify genes and pathways in the primary breast tumor that are tightly linked to genes and pathways in the patient blood cells.[11] The web application interfaces with the methods implemented as an R package and integrates the results together with information from biological databases through a simple user interface.

A third application we developed was a simple re-deployment of the MIXT web application with a new dataset. In this application we simply replaced the R package with a new package that interfaced with different data. All the other components are reused and highlights the flexibility of the approach.

Combined these systems and applications demonstrate how small modular entities are useful for both batch processing of datasets, as well as interactive applications.

1.4 Summary of Results

We show the viability of our approach through real-world applications in systems epidemiology and precision medicine. We demonstrate its usefulness for building interactive data exploration application, implemented in Kvik. We show the applicability of small modular entities in deep analysis pipelines, as implemented in walrus.

We have used walrus to analyze a whole-exome dataset to from a sample in the McGill Genome Quebec [MGGQ] dataset (GSE58644)[12] to discover Single Nucleotide Polymorphisms (SNPs), genomic variants and somatic mutations. Using walrus to analyze a dataset added 10% to the runtime and doubled the space requirements, but reduced days of compute time down to seconds when restoring a previous pipeline configuration.

We have used the packages in Kvik to develop a web application, MIxT blood-tumor, for exploring and comparing transcriptional profiles from blood and tumor samples in breast cancer patients. In addition we have used it to build an application to explore gene expression data in the context of biological pathways. We show that developing an application using a microservice approach allows us to reduce database query times down to 90%, and that we can provide an interface to statistical analyses that is up to 10 times as fast as alternative approaches.

Together the results show that our approach, small modular entities, can be used to enable reproducible data analysis and exploration of high-throughput biological datasets while still providing the required performance.

1.5 List of papers

This section contains a list of papers along with short descriptions and my contributions to each paper.

Paper 1

Title	Kvik: three-tier data exploration tools for flexible analysis of genomic data in epidemiological studies
Authors	**Bjørn Fjukstad**, Karina Standahl Olsen, Mie Jareid, Eiliv Lund, and Lars Ailo Bongo
Description	The initial description of Kvik, and how we used it to implement Kvik Pathways, a web application for browsing biologicap pathway maps integrated with gene expression data from the NOWAC cohort.
Contribution	I designed, implemented, and deployed Kvik and Kvik Pathways. Evaluated the system and wrote the manuscript.
Publication date	15 March 2015
Publication venue	F1000
Citation	[9] B. Fjukstad, K. S. Olsen, M. Jareid, E. Lund, and L. A. Bongo, "Kvik: three-tier data exploration tools for flexible analysis of genomic data in epidemiological studies," *F1000Research*, vol. 4, 2015

Paper 2

Title	Building Applications For Interactive Data Exploration In Systems Biology.
Authors	**Bjørn Fjukstad**, Vanessa Dumeaux, Karina Standahl Olsen, Michael Hallett, Eiliv Lund, and Lars Ailo Bongo.
Description	Describes how we further developed the ideas from Paper 1 into an approach that we used to build the MIXT web application.
Contribution	Designed, implemented, and deployed Kvik and the MIXT web application. Evaluated the system and wrote the manuscript.
Publication date	20 August 2017.
Publication venue	The 8th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (ACM BCB) August 20–23, 2017.
Citation	[8] B. Fjukstad, V. Dumeaux, K. S. Olsen, E. Lund, M. Hallett, and L. A. Bongo, "Building applications for interactive data exploration in systems biology," in *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*. ACM, 2017, pp. 556–561

Paper 3

Title	Interactions Between the Tumor and the Blood Systemic Response of Breast Cancer Patients
Authors	Vanessa Dumeaux, **Bjørn Fjukstad**, Hans E Fjosne, Jan-Ole Frantzen, Marit Muri Holmen, Enno Rodegerdts, Ellen Schlichting, Anne-Lise Børresen-Dale, Lars Ailo Bongo, Eiliv Lund, Michael Hallett.
Description	Describes the MIXT system which enables identification of genes and pathways in the primary tumor that are tightly linked to genes and pathways in the patient Systemic Response (SR).
Contribution	Designed, implemented, and deployed the MIXT web application. Contributed to write the manuscript.
Publication date	28 September 2017.
Publication venue	PLoS Computational Biology
Citation	[11] V. Dumeaux, B. Fjukstad, H. E. Fjosne, J.-O. Frantzen, M. M. Holmen, E. Rodegerdts, E. Schlichting, A.-L. Børresen-Dale, L. A. Bongo, E. Lund *et al.*, "Interactions between the tumor and the blood systemic response of breast cancer patients," *PLoS Computational Biology*, vol. 13, no. 9, p. e1005680, 2017

Paper 4

Title	A Review of Scalable Bioinformatics Pipelines
Authors	**Bjørn Fjukstad**, Lars Ailo Bongo.
Description	This review survey several scalable bioinformatics pipelines and compare their design and their use of underlying frameworks and infrastructures.
Contribution	Literature review and Wrote the manuscript.
Publication date	23 October 2017
Publication venue	Data Science and Engineering 2017.
Citation	[13] B. Fjukstad and L. A. Bongo, "A review of scalable bioinformatics pipelines," *Data Science and Engineering*, vol. 2, no. 3, pp. 245–251, 2017

Paper 5

Title	nsroot: Minimalist Process Isolation Tool Implemented With Linux Namespaces.
Authors	Inge Alexander Raknes, **Bjørn Fjukstad**, Lars Ailo Bongo.
Description	Describes a tool for process isolation built using Linux namespaces.
Contribution	Contributed to the manuscript, specifically to the literature review and related works.
Publication date	26 November 2017
Publication venue	Norsk Informatikkonferanse 2017.
Citation	[13] B. Fjukstad and L. A. Bongo, "A review of scalable bioinformatics pipelines," *Data Science and Engineering*, vol. 2, no. 3, pp. 245–251, 2017

Paper 6

Title	Transcription factor PAX6 as a novel prognostic factor and putative tumour suppressor in non-small cell lung cancer
Authors	Yury Kiselev, Sigve Andersen, Charles Johannessen, **Bjørn Fjukstad**, Karina Standahl Olsen, Helge Stenvold, Samer Al-Saad, Tom Dønnem, Elin Richardsen, Roy M Bremnes, and Lill-Tove Rasmussen Busund.
Description	This paper explores the possibility of using the PAX6 transcription factor as a prognostic marker in non-small cell lung cancer.
Contribution	Did the analyses to explore association between PAX6 gene expression and PAX6 target genes.
Publication date	22 March 2018
Publication venue	Scientific Reports 2018.
Citation	[14] Y. Kiselev, S. Andersen, C. Johannessen, B. Fjukstad, K. S. Olsen, H. Stenvold, S. Al-Saad, T. Donnem, E. Richardsen, R. M. Bremnes *et al.*, "Transcription factor pax6 as a novel prognostic factor and putative tumour suppressor in non-small cell lung cancer," *Scientific reports*, vol. 8, no. 1, p. 5059, 2018

Paper 7

Title	Reproducible Data Analysis Pipelines in Precision Medicine
Authors	**Bjørn Fjukstad**, Vanessa Dumeaux, Michael Hallett, Lars Ailo Bongo
Description	This paper outlines how we used the container centric development model to build walrus.
Contribution	Design, implementation and evaluation of walrus. Wrote the manuscript.
Publication date	TBA
Publication venue	TBA
Citation	[?]

1.6 Dissertation Plan

This thesis is organized as follows. Chapter 2 describes the characteristics of state-of-the-art biological datasets in systems epidemiology and how we have developed an approach to analyze these. In Chapter 3 we describe how we used the same ideas and model to develop applications for interactively exploring results from statistical analyses. Chapter 4 explores how we can develop analysis pipelines for high-throughput sequencing datasets in precision medicine. It describes in detail how we use a container centric development model to build a tool, walrus, to develop and execute these pipelines. Finally, Chapter 5 concludes the work and discusses future directions.

2

Modern Biological Data Analysis

From the discovery of the DNA structure by Watson and Crick in 1953[15] to the sequencing of the human genome in 2001 [16, 17] and the massively parallel sequencing platforms in the later years[6], the scientific advances have been tremendous. Today, single week-long sequencing runs can produce as much data as did entire genome centers just years ago.[18] These technologies allow researchers to produce data faster, cheaper and more efficiently, now making it possible to sequence the entire genome from a patient in less than a days work.

In this chapter we give a background in the different aspects of analyzing and exploring biological datasets. We use the NOWAC study as an example and highlight the necessary processing steps from data generation and to interpretation of results. In addition we describe the traditional data analysis and management, and propose a novel approach for organizing, sharing, and collaborating on research data and analyses. In this chapter we focus on analysts in the NOWAC study writing their own specialized analysis scripts in the R programming language.

2.1 High-Throughput Datasets for Research and Clinical Use

Cells are the smallest units an organism can be divided into, that still possesses the functions performed by living organisms. Within cells we find proteins, the working units, found in a wide range of processes. All cells within an organism contain the same genetic information, this genetic information is stored within nucleic acids which are responsible for storage, transmission and expression. There are two types of nucleic acids, DNA and RNA. DNA is responsible for the storage of genetic information, while RNA is used in decoding the information stored within DNA. Genes are sequences of DNA, and the human genome consists of approximately 20 500 genes. These genes specify how proteins are synthesized. In short, DNA is transcribed into RNA which are translated to proteins. This is called the central dogma of molecular biology.

DNA sequencing is the process of determining the order of nucleotides within a strand of DNA. High-throughput Sequencing (HTS), or Next-generation Sequencing (NGS), is a term used to described newer technology that enables massively-parallel sequencing of DNA. HTS instruments sequence millions of short base pairs and we assemble these in the data analysis process. Typical sequencing datasets are in the size of hundreds of Gigabytes (GBs) per sample.

While HTS can study the sequence of bases, we use DNA microarrays to study the transcriptome, or the genes actively expressed. While the genome is mostly fixed for an organism, the transcriptome is continuously changing. These instruments report the expression levels of a large number of target genes, and by profiling these we can study which genes are active in the biological sample. Microarray datasets are in the size of of megabytes per sample.

Another technique to study the transcriptome is to use RNA-seq technology based on HTS. RNA-seq instruments also read millions of short base pairs in parallel, and can be used in gene expression analysis. Because of its higher quality output, RNA-seq is the successor to microarray technology. These datasets are also in the size of hundreds of GBs.

Precision medicine uses patient-specific molecular information to diagnose and categorize disease to tailor treatment to improve health outcome.[19] Important research goal in precision medicine are to learn about the variability of the molecular characteristics of individual tumors, their relationship to outcome, and to improve diagnosis and therapy.[20] International cancer institutions are therefore offering dedicated personalized medicine programs, but while the data collection and analysis technology is emerging, there are still unsolved

problems to enable reproducible analyses in clinical settings. For cancer, HTS is the main technology to facilitate personalized diagnosis and treatment since it enables collecting high quality genomic data from patients at a low cost.

2.2 Norwegian Women and Cancer (NOWAC)

The NOWAC study is a prospective population-based cohort that tracks 34% of all Norwegian women born between 1943–57.[?] We started the data collection in NOWAC in 1991 with questionnaires to cover, among others, the use of oral contraceptives and hormonal replacement therapy, reproductive history, smoking, physical activity, breast cancer, and breast cancer in the family. We also integrate the Norwegian Cancer Registry, the register of the National Mammographic Screening Program, and the register of death certificates in Statistics Norway. In addition to the questionnaire data, the NOWAC biobank now contain blood samples from 50 000 women, as well as tumor tissue samples. From the biological samples we have generated gene expression, miRNA, methylation, and RNA sequencing datasets.

From the NOWAC cohort we have have published a number of research papers that investigate the questionnaire data[?]. We have also used the gene expression datasets to explore ... and interactions between the tumor and the blood systemic response of breast cancer patients.[11] While we have studied interesting patterns and questions, there are still unexplored areas in the available datasets.

2.2.1 Data Management

In the NOWAC study questionnaire data is stored on an in-house dedicated database backed up to an independent storage node. The database is maintained by a handful of personnel which are also responsible for extracting data for researchers and projects. This is typically done through SAS scripts that selects the applicable variables and samples, and optionally generate computed variables such as smoking status from the questionnaire data. There is no systematic version control of the datasets or processing scripts.

In addition to the questionnaire data, the NOWAC study also integrates with registries which are updated regularly. The datasets from the different registries are typically delivered as Comma-separated values (CSV) files which are then processed into a standardized format. Since the NOWAC study is a prospective cohort women are expected to get cancer and move from the list of controls into the list of cases.

In the NOWAC study we have used third-party sources to process and analyze biological samples. The resulting datasets were then stored on a local compute-node and made available to researchers on demand. Because of the nature of the biological datasets, many of these require extensive pre-processing before they are analysis-ready.

There are multiple drawbacks to managing the research datasets using the traditional approach detailed above. First, it is a difficult task to retrace original data when the data extraction scripts are not versioned and kept track of in a shared repository. Second, while datasets are backed up, there is no information about changes between dataset versions. For example, samples may be removed and there is no record of these changes.

2.2.2 Data Analysis

Traditionally, researchers had to send an application to get data exported from the database by an engineer. The downstream analysis of questionnaire data was typically done in SAS on local computers. Traditionally researchers have used e-mail to communicate and share data analysis scripts, and there has not been a central hub to share scripts or data. Because of the many packages in Bioconductor[1] we have used R to analyse the different gene expression datasets in the projcet. However like with SAS the researchers have typically shared scripts through e-mail and there has not been a tradition to version control the analysis scripts. Results from the analyses are often communicated through research papers subsequently.

There are several drawbacks to this approach. First, there are obvious drawbacks of not version controlling analysis scripts Second, sharing analysis scripts through e-mail and not centralized repositories makes it difficult to share code especially in research groups with researcher turnover. Third, using proprietary software such as SAS makes it difficult for anyone without the necessary licences to revisit the analyses. Last, separating the reporting of the results from the actual data analysis leaves room for human errors when generating tables or plots.

2.2.3 Requirement Analysis

To enable more researchers to benefit from the unique datasets in NOWAC we wanted to improve the way researchers access data as well as how they share their analyses. By improving access and how the researchers share analyses

1. bioconductor.org

we are indirectly improving reproducibility in the project.

Before developing an approach to standardize data management and analysis in the NOWAC study we performed a requirement analysis and determined the following requirements for such systems:

- It provides a single location for storing datasets and analysis code.
- it provides documentation for the datasets, how they are processed, and other useful information for downstream analyses.
- It enables the standardization of pre-processing steps required to analyze biological datasets.

From these requirements we designed and implemented i) a software package in the R programming language that contain all datasets, their documentation, and useful analysis functions to work with the data, and ii) a GUI for to perform the necessary pre-processing steps before delivering datasets to researchers, and iii) best practices for researchers that want to work with datasets from the NOWAC cohort. In the next sections we describe the software packages and best practices.

2.3 Modernizing Data Management and Analysis

The first step in modernizing the data management and analysis in the NOWAC study was to identify the possible environments to develop our approach. While the majority of the researchers that have worked on the project have used SAS for their analyses of the questionnaire data, all researchers working on biological data are using R. Because of the number of additional software packages, its open-source implantation, and growing developer community we have opted to implement our approach for managing and processing to fit the R programming language.

The great strength of R comes from its many packages for analyzing, plotting, and interpreting data. An R package consists of a code, documentation, tests, and data. Bioconductor and CRAN provide online hosing for a large number of packages, and users can mix and match these packages to fit their need.

We designed the NOWAC R package to provide a single-stop location for documentation, data, and analysis code for researchers within our research group. We share this package within our group both as a repository that researchers can contribute to, and a binary installer for basic uses. We use `git` to ver-

sion control the analysis code and datasets, and store the repository on a self-hosted git server (gitlab[2]). With git we can track changes to our analysis code, documentation, and datsets over time.

2.3.1 Data management

We bundle together all datasets with the NOWAC package. This includes both questionnaire, registry, and biological datasets. Since none of these are perticularly large in size (no single dataset being more than tens of GBs) we are able to distribute them with the software package. Some of the datasets require pre-processing steps such as outlier removal before the analysts can explore the datasets. For these datasets we store both the *raw* datasets as well as the analysis-ready clean datasets. We store the raw datasets in their original format, while clean datasets are stored as R data files to simplify importing them in R. In addition to the datasets themselves we store the R code we used to generate the datasets. For clearity we decorate the scripts with specially formatted comments that can be used with knitr[3] to generate PDF reports. These highlight the transformation of the data from raw to clean, with information such as removed samples or data normalization methods.

We document both code and each dataset with roxygen2[4]. The documentation is written as specially formatted comments to R source code and allows us to special details such as data collection date, instrument types, the persons involved with data collection and analysis, pre-processing methods etc. When users install the NOWAC package these comments are used to generate interactive help pages which they can browse in R, be it the CLI or through RStudio.

As mentioned we use git to version control our NOWAC package. There are however drawbacks to creating one large repository for both data and code. Since git stores every version of a file, these types of repositories may become large if the datasets are changing a lot over time, and are stored in binary formats, e.g. gene expression datasets. We have explored different techniques to minimize our repository and have opted to store all datasets as git submodules[5]. Submodules allow us to keep the main repository size down while still versioning the data. Other alternatives such as git-raw[6], git-annex[7] git-lfs[8]

2. `gitlab.com`
3. `yihui.name/knitr`
4. `cran.r-project.org/web/packages/roxygen2`
5. `git-scm.com/docs/git-submodule`
6. `github.com/atofigh/git-raw`
7. `git-annex.branchable.com`
8. `git-lfs.github.com`

exist, and these all provide alternative approaches to storing large binary files in the repository. We have not found that any of these could satisfy our needs, mainly because they all require a newer software stack than we have access to.

2.3.2 Data processing

In the NOWAC package we provide utility functions to get started with the analysis of our datasets. Because of the specialized nature of the different research project the NOWAC package only contains helper functions to start analyzing NOWAC data, e.g. retrieving questionnaire data.

2.3.3 Best Practices

From our experiences we have developed a set of best practices for researchers working on data analysis in the NOWAC study. We believe that we can generalize these to researchers working in different disciplines.

Document every step in the analysis. Analysis of modern datasets is a complex exercise with the possibility of introducing an error in every step. Analysts often use different tools and systems that require a particular set of input parameters to produce results. Thoroughly document every step from raw data to the final tables that go into a manuscript.

In the NOWAC study we write help pages and reports for all datasets, and the optional pre-processing steps.

Generate reports and papers using code. With tools such as R Markdown[9] and kntir there are few reasons for decoupling analysis code with the presentation of the results through reports or scientific papers. Doing so ensures the correctnes reported results from the analyses, and greatly simplifies reproducing the results in a scientific paper.

In the NOWAC study we produce reports from R code. These include preprocessing and data delivery of datasets to researchers. One example of a report is the analyses done in [14] where we documented the association between PAX6 gene expression and PAX6 target genes.

Version control everything. Both code and data changes over the course of a research project. Version control everything to make it possible to retrace

9. `rmarkdown.rstudio.com`

changes and the person responsible for them. It is often necessary to roll back to previous versions or a dataset or analysis code, or to identify the researches that worked on specific analyses.

In the NOWAC study we promote the use of git to version control both source code and data.

Collaborate and share code through source code management (SCM) systems. Traditional communication through e-mail makes it difficult to keep track of existing analyses and their design choices both for existing project members and new researchers. With SCM hosting systems such as Github developing analysis code becomes more transparent to other collaborators, and encourages collaboration. It also simplifies the process or archiving development decisions such as choosing a normalization method.

In the NOWAC study we collaborate on data analysis through a self-hosted Gitlab[10] installation. We also open-source our code on Github.

2.4 Pippeline

The Pippeline is our approach to standardize the pre-processing and generation of analysis-ready datasets in the NOWAC study. It is a point-and-click application that can be used by users without any prior programming experience. The Pippeline application builds upon the NOWAC R package, both fundamentally but also through its design choices. It is targeted towards researchers who want to start analyzing different datasets from the NOWAC cohort, and provides an automated system for performing the first steps of such analyzes. Figure 2.1 shows a screenshot of the web application.

The Pippeline is implemented as an interactive web application, and users click through a series of steps before receiving an analysis-ready datasets. Users specify details such as what type of study design, e.g. cross-sectional or case-control, the data normalization methods, and integration with the questionnaires.

The Pippeline is implemented as an interactive web application written in R using the `Shiny` framework. Besides the `shiny` framework, the Pippeline uses several handful packages for the data analysis, such as `lumi`, `limma`, `sva`, `genefilter`, `nlme` and `Biobase`. These packages provide tools for managing our datasets, methods for processing and analysis.

10. `gitlab.com`

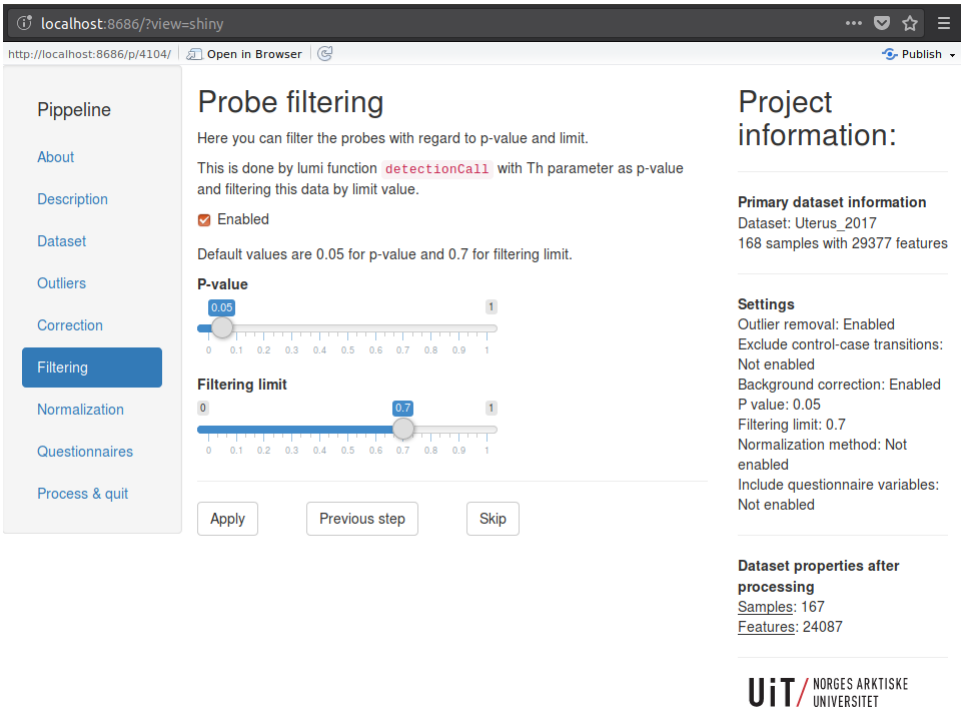


Figure 2.1: The Pippeline web application. The screenshot shows the probe filtering steps of preparing a gene expression dataset.

3

Interactive Exploration

Explorative analysis is essential for understanding biological functions in large-scale omics' datasets. Applications with interactive visualizations and interfaces can help researchers study the datasets to discover emerging patterns. Integrating omics' data from large epidemiological studies requires collecting samples from thousands of people at different biological levels over a long period of time. It is therefore usual to reuse the data from an epidemidemiological cohort to answer different research questions using different study designs. Although an existing tool may be useful for one project, no tool provides the required functionality for several different projects. The cost of data collection has drastically decreased, but data analysis continue to represent a large fraction of the total cost of these studies.[21]

The main goal of a data exploration application in bioinformatics is to help users discover interesting patterns in a biological dataset. Because of the complexity of biological data and analyses, we need specialized software to help find these patterns. Such software is usually written in statistical programming languages, such as R, and often do not provide interfaces outside these programming environments requiring application developers to implement their applications within these environments. We believe that it is possible to implement an interface to specialized statistical software that allows application developers to build exploration tools using the most suitable environment and technology.

To interpret data, experts regularly exploit prior knowledge via database queries

and the primary scientific literature. There are a wealth of online databases, some of which provide open APIs in addition to web user interfaces that application developers can make use of. While the databases can provide helpful information, there are some limitations associated with their integration into interactive data exploration applications: i) the APIs are not fast enough to use in interactive applications where the application has to perform multiple database queries, ii) some databases put restrictions on the number of database queries, and iii) there is no uniform way for storing additional database metadata to identify database versions and query parameters.

Through the development of several data exploration applications, we have iteratively developed an approach to build such applications. These applications have in common a set of features In this chapter we discuss how we can build data exploration applications using the SME approach through its implementation in Kvik. We demonstrate the usefulness of the approach through a set of different applications for exploring transcriptional profiles from the NOWAC cohort. While these applications provide specialized user interfaces, we show how the design patterns and ideas can be used in a wide range of use cases.

Data exploration tools tend to specialize on a handful of analysis tasks, but while the applications themselves are specific they often share the same underlying components. Both Kvik Pathwyas and the MIXT web applications interface with the same online databases, and rely on statistical analyses to provide users with data to explore. Designing systems as collections of smaller components allow developers to reuse parts and shortens development time.

3.1 Motivating Examples

The need for interactive applications has come from two different studies within the NOWAC project. Both of these rely on advanced statistical analyses and produce comprehensive results that are interpreted by researchers through integration with online databases and interactive visualizations. The end results are typically large tables that require manual inspection. Below we describe the two applications before we go into the requirements, design and implementation of the applications.

3.1.1 High and Low Plasma Ratios of Essential Fatty Acids

The aim of the first application was a to explore the results from a previous published project ([10], doi: 10.1371/journal.pone.0067270) that compared gene

expression in blood from healthy women with high and low plasma ratios of essential fatty acids. Gene expression differences where assessed and determined that there were 184 differentially expressed genes. When exploring this list of 184 genes, functional information was retrieved from GeneCards and other repositories, and the list was analyzed for overlap with known pathways using MSigDB (available online at broadinstitute.org/gsea/msigdb). The researchers had to manually maintain overview of single genes, gene networks or pathways, and gather functional information gene by gene while assessing differences in gene expression levels. With this approach, researchers were limited by their own capacity to retrieve information manually from databases and keep it up to date. An application could automate the retrieval and ensure that the data is correct and up to date.

3.1.2 Matched Interactions Across Tissues (MIxT)

The aim of the Matched Interactions Across Tissues (MIxT) study was to identify genes and pathways in the primary breast tumor that are tightly linked to genes and pathways in the patient blood cells.[11] We generated and analyzed expression profiles from blood and matched tumor cells in 173 breast cancer patients included in the Norwegian Women and Cancer (NOWAC) study. The MIxT analysis starts by identifying sets of genes tightly co-expressed across all patients in each tissue. Each group of genes or modules were annotated based on a priori biological knowledge about gene functionality. Then the analyses investigate the relationships between tissues by asking if specific biologies in one tissue are linked with (possibly distinct) biologies in the second tissue, and this within different subgroup of patients (i.e. subtypes of breast cancer).

3.2 Requirements

From these two studies we identified a set of requirements that the data exploration applications should satisfy:

Interactive The applications should provide interactive exploration of datasets through visualizations and integration with relevant information. To understand the large quantities of heterogeneous data in epidemiological studies, researchers need interactive visualizations that provide different views and presentations of the data. Also, to understand the results it is important to have instant access to existing knowledge from online databases.

Familiar They should use familiar visual representations to present informa-

tion to researchers. For more efficient data exploration it is effective to use representations that researchers are familiar with both from the literature and from other applications.

Simple to use Researchers should not need to install software to explore their data through the applications. The applications should protect the researcher from the burden of installing and keeping an application up to date.

Lightweight Data presentation and computation should be separated to make it possible for researchers to explore data without having to have the computational power to run the analyses. With the growing rate data is produced at, we cannot expect that researchers have the resources to store and analyze data on their own computers.

With these requirements in mind we set out to develop two applications for interactively explore the results from the studies along with information from online databases.

3.3 Kvik Pathways

The first application we developed was Kvik Pathways. Kvik Pathways allows users to interactively explore a molecular dataset, such as gene expression, through a web application. It provides pathway visualizations and detailed information about genes and pathways from the KEGG database. 3.1 Through pathway visualizations and integration with the KEGG databases, users can perform targeted exploration of pathways and genes to get an overview of the biological functions that are involved with gene expression from the underlying dataset. Kvik Pathways gathers information about related pathways and retrieves relevant information about genes, making it unnecessary for researchers to spend valuable time looking up this information manually. For example, navigating a set of pathways and browsing information about genes in these, requires the researcher to manually query KEGG for each specific gene. Kvik Pathways retrieves information about genes without the researcher having to leave the pathway visualization to retrieve relevant information.

3.3.1 Analysis Tasks

To efficiently develop the application we designed 3 analysis tasks that the application supports.

A1: Explore gene expression in the context of Kyoto Encyclopedia of Genes and Genomes (KEGG) pathway maps. It provides users with a list of pathway maps to choose from, and the application will generate an interactive visualization including gene epxression values.

A2: Investigate and retrieve relevant biological information. It provides users with direct links to online databases with up to date information.

A3: Explore relationships between pathway maps. When users select a gene from a pathway map they get a list of other pathway maps that this gene is found in, in addition to their similarity. This allows users to dig into the different processes genes are a part of.

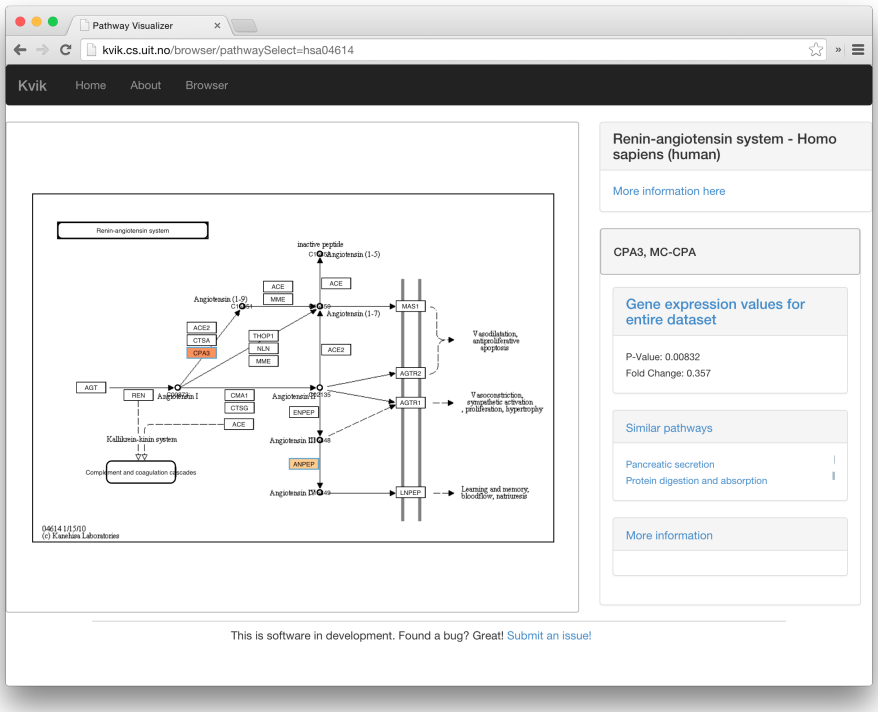


Figure 3.1: Screenshot of the renin-angiotensin pathway (KEGG pathway id hsa04614) in Kvik Pathways. The user has selected the gene CPA3, which brings up the panel on the right. From here researchers can browse pathways that the gene is a member of, and read relevant information about the gene from KEGG

3.3.2 Architecture

Kvik Pathways has a three-tiered architecture of independent layers (Figure 3.2). The browser layer consists of the web application for exploring gene expression data and biological pathways. A front-end layer provides static content such as HTML pages and stylesheets, as well as an interface to the data sources with dynamic content such as gene expression data or pathway maps to the web application. The back-end layer contains information about pathways and genes, as well as computational and storage resources to process genomic data such as the NOWAC data repository. The Kvik framework provides the components in the back-end layer.

The Data Engine in the back-end layer provides an interface to the NOWAC data repository stored on a secure server on our local supercomputer. In Kvik Pathways all gene expression data is stored on the computer that runs the Data Engine. The Data Engine runs an R session accessible over remote procedure calls (RPCs) from the front-end layer using RPy2 (rpy.sourceforge.net) to interface with R. To access data and run analyses the Data Interface exposes a HTTP API to the browser layer (Table 1 provides the interfaces).

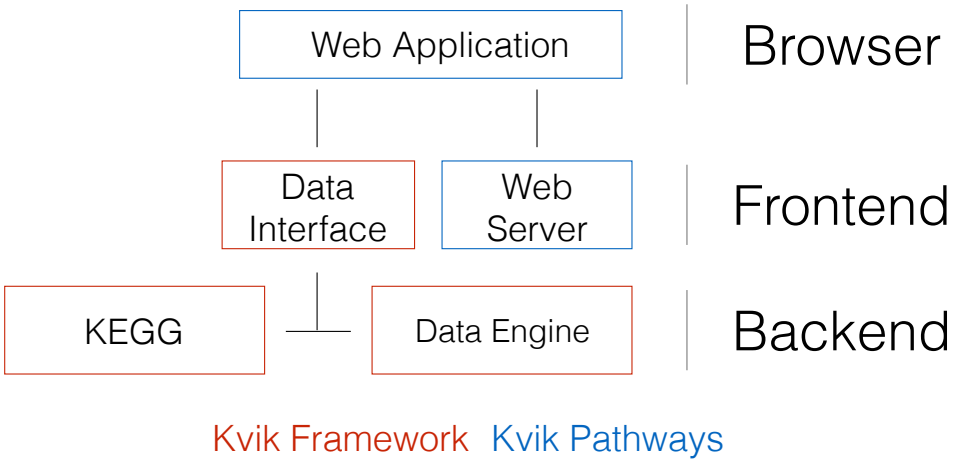


Figure 3.2: The three-tiered architecture of Kvik Pathways.

3.3.3 Implementation

To create pathway visualizations the Kvik back-end retrieves and parses the KEGG Markup Language (KGML) representation and pathway image from KEGG databases through its REST API (rest.kegg.jp). This KGML representation of a pathway is an XML file that contains a list of nodes (genes, proteins or compounds) and edges (reactions or relations). Kvik parses this file and

Table 3.1: The REST interface to the Data Engine, for example, use `/genes/` to retrieve all available genes in our dataset.

URL	Description
/fc/[genes...]	Calculate and retrieve fold-change for the specified genes
/pvalues/[genes...]	Calculate and retrieve *p*-values for the specified genes
/exprs/[genes...]	Get the raw gene expression values from the dataset
/genes	Get a list of all genes in the dataset

generates a JSON representation that Kvik Pathway uses to create pathway visualizations. Kvik Pathways Cytoscape.js to create a pathway visualization from the list of nodes and edges and overlay the nodes on the pathway image. To reduce latency when using the KEGG REST API, we cache every response on our servers. We use the average fold change between the groups (women with high or low plasma ratios of essential fatty acids) in the dataset to color the genes within the pathway maps. To highlight *p*-values, the pathway visualization shows an additional colored frame around genes. We visualize fold change values for individual samples as a bar chart in a side panel. This bar chart gives researchers a global view of the fold change in the entire dataset.

Kvik provides a flexible statistics back-end where researchers can specify the analyses they want to run to generate data for later visualization. For example, in Kvik Pathways we retrieve fold change for single genes every time a pathway is viewed in the application. These analyses are run ad hoc on the back-end servers and generates output that is displayed in the pathways in the client's web browser. The data analyses are implemented in an R script and can make use of all available libraries in R, such as Bioconductor (bioconductor.org).

Researchers modify this R script to, for example, select a normalization method, or to tune the false discovery rate (FDR) used to adjust the *p*-values that Kvik Pathways uses to highlight significantly differentially expressed genes. Since Kvik Pathways is implemented as a web application and the analyses are run ad hoc, when the analyses change, researchers get an updated application by simply refreshing the Kvik Pathways webpage.

3.3.4 Practical Use

As an example of practical use of Kvik Pathways, we chose one of the significant pathways from the overlap analysis, the renin-angiotensin pathway (Supplementary table S5 in [10]). The pathway contains 17 genes, and in the pathway map we could instantly identify the two genes that drive this result. The color of the gene nodes in the pathway map indicates the fold change, and

the statistical significance level is indicated by the color of the node's frame. We use this image of a biological process to see how these two genes (and their expression levels) are related to other genes in that pathway, giving a biologically more meaningful context as compared to merely seeing the two genes on a list.

3.4 Design Principles

Through the experiences developing the Kvik Pathways we identified a set of components and features that are central to building data exploration applications:

1. A low-latency language-independent approach for integrating, or embedding, statistical software, such as R, directly from a data exploration application.

2. A low-latency language-independent interface to online reference databases in biology that users can query to explore results in context of results in context of known biology.

3. A simple method for deploying and sharing the components of an application between projects.

In the following sections we describe how we designed and implemented the packages in Kvik, and how they later formed the bases of the SME approach that the MIXT web application builds upon.

3.5 Kvik

Kvik is a collection of software packages in the Go programming language that allows developers to build data exploration applications. It is the basis of our two data exploration applications, and has been iteratively developed through the last years. Kvik provides an interface to the R statistical programming language, both as a stand-alone service, a client library, and through an OpenCPU server. It provides an R-based pipelining tool that allows useres to specify and run statistical analysis pipelines in R. Kvik also contains a Javascript package for visualizing KEGG pathways using d3. In addition it provides an interface with online databases such as MsigDB and KEGG.

We used the experience building Kvik Pathways to completely re-design and re-

implement the R interface in Kvik. From having an R server that can run a set of functions from an R script, it now has a clean interface to call any function from any R package, not just retrieving data as a text string but in a wide range of formats. We also re-built the database interface, which is now a separate service. This makes it possible to leverage its caching capabilities to improve latency. This transformed the application from being a single monolithic application into a system that consists of a web application for visualizing biological pathways, a database service to retrieve pathway images and other metadata, and a compute service for interfacing with the gene expression data in the NOWAC cohort. We could then re-use the database and the compute service in the MIxT application.

We have used these packages to develop the SME approach through services that provide open interfaces to the R programming language and the online databases. We outline these services in 3.5.1. In short the interfaces are accessible through an HTTP interface and can be used from any platform.

3.5.1 Microservices

We generalized our efforts from Kvik Pathways into the following design principles for building applications in bioinformatics:

Principle 1: Build applications as collections of language-agnostic microservices. This enables re-use of components and does not enforce any specific programming language on the user interfaces or the underlying components of the application.

Principle 2: Use software containers to package each service. This has a number of benefits: it simplifies deployment, ensures that dependencies and libraries are installed, and simplifies sharing of services between developers.

Compute Service

We have built a compute service that provides an open interface directly to the R programming language, thus providing access to a wealth of algorithm and statistical analysis packages that exists within the R ecosystem. Application developers can use the compute service to execute specialized analyses and retrieve results either as plain text or binary data such as plots. By interfacing directly with R, developers can modify input parameters to statistical methods directly from the user-facing application.

The compute service offers three main operations to interface with R: i) to call

a function with one or more input parameters from an R package, ii) to get the results from a previous function call, and iii) a catch-all term that both calls a function and returns the results. We use the same terminology as OpenCPU[22] and have named the three operations Call, Get, and RPC respectively. These three operations provide the necessary interface for applications to include statistical analyses in the applications.

The compute service is implemented as an HTTP server that communicates with a pre-set number of R processes to execute statistical analyses. At initiation of the compute service, a user-defined number of R worker sessions are launched for executing analyses (default is 5). The compute service uses a round-robin scheduling scheme to distribute incoming requests to the workers. We provide a simple FIFO queue for queuing of requests. The compute service also provides the opportunity for applications to cache analysis results to speed up subsequent calls.

Database Service

To alleviate application developers of the challenges in 3.4, we built an database service that provides a solution to the three. The service provides low latency, minimizes the number of queries to remote databases, and stores additional metadata to capture query parameters and database information. The database service provides an open HTTP interface to biological databases for retrieving meta-data on genes and processes. We currently have packages for interfacing with E-utilities,[1] MSigDB, HGNC, and KEGG.

Both the compute and the databases service in Kvik build on the standard *net/http* package in the Go programming language.[2] The database service use the *gocache*[3] package to cache any query to an online database. In addition we deploy each service as Docker containers.[4]

3.6 MIXT

The MIXT system is an online web application for exploring and comparing transcriptional profiles from blood and tumor samples. It provides users with an interface to explore high-throughput gene expression profiles of breast cancer

1. `eutils.ncbi.nlm.nih.gov`.
2. `golang.org`
3. `github.com/fjukstad/gocache`.
4. Available at `hub.docker.com/r/fjukstad/kvik-r` and `hub.docker.com/r/fjukstad/db`.

tumor data with matched profiles from the patients blood.

3.6.1 Analysis Tasks

To efficiently develop the application we defined six analysis tasks (A1-A6) that the application supports:

A1: Explore co-expression gene sets in tumor and blood tissue. Users can explore gene expression patterns together with clinicopathological variables (e.g. patient or tumor grade, stage, age) for each module. In addition we enable users to study the underlying biological functions of each module by including gene set analyses between the module genes and known gene sets.

A2: Explore co-expression relationships between genes. Users can explore the co-expression relationship as a graph visualization. Here genes are represented in the network with nodes and edges represent statistically significant correlation in expression between the two end-points.

A3: Explore relationships between modules from each tissue. We provide two different metrics to compare modules, and the web application enables users to interactively browse these relationships. In addition to providing visualizations the compare modules from each tissue, users can explore the relationships, but for different breast cancer patient groups.

A4: Explore relationships between clinical variables and modules. In addition to comparing the association between modules from both tissues, users also have the possibility to explore the association with a module and a specific clinical variable. It is also possible to explore the associations after first stratifying the tumors by breast cancer subtype (an operation that is common in cancer related studies to deal with molecular heterogeneity).

A5: Explore association between user-submitted gene lists and computed modules. We want to enable users to explore their own gene lists to explore them in context of the co-expression gene sets. The web application must handle uploads of gene lists and compute association between the gene list and the MIxT modules on demand.

A6: Search for genes or gene lists of interest. To facilitate faster lookup of genes and biological processes, the web application provides a search functionality that lets users locate genes or gene lists and show association to the co-expression gene sets.

3.6.2 Architecture

We structured the MIxT application with a separate view for each analysis task. To explore the co-expression gene sets (**A1**), we built a view that combines both static visualizations from R together with interactive tables for gene overlap analyses. Figure 3.3 shows the web page presented to users when they access the co-expression gene set 'darkturquoise' from blood. To explore the co-expression relationship between genes (**A2**) we use an interactive graph visualization build with Sigmajs[5]. We have built visualization for both tissues, with graph sizes of 2705 nodes and 90 348 edges for the blood network, and 2066 nodes and 50 563 edges for the biopsy network. To visualize relationships between modules from different tissues (**A3**), or their relationship to clinical variables (**A4**) we built a heatmap visualization. We built a simple upload page where users can specify their gene sets (**A5**). The file is uploaded to the web application which redirects it to a backend service that runs the analyses. Similarly we can take user input to search for genes and processes (**A6**).

Figure 3.3: MIxT module overview page. The top left panel contains the gene expression heatmap for the module genes. The top right panel contains a table of the genes found in the module. The bottom panel contains the results of gene overlap analyses from the module genes and known gene sets from MSigDB.

5. `sigmajs.org`.

For the original analyses we built an R package, mixtR,[6] with the statistical methods and static visualizations for identifying associations between modules across tissues. To make the results more easily accessible we built a web application that interfaces with the R package, but also online databases to retrieve relevant metadata. To make it possible to easily update or re-implement parts of the system without effecting the entire application, and we developed it using a microservice architecture. The software containers allowed the application to be deployed on a wide range of hardware, from local installations to cloud systems.

3.6.3 Implementation

From the six analysis tasks we designed and implemented MIxT as a web application that integrates statistical analyses and information from biological databases together with interactive visualizations. Figure 3.4 shows the system architecture of MIxT which consists of three parts i) the web application itself containing the user-interface and visualizations; ii) the compute service performing the MIxT analyses developed in an R package, delivering data to the web application; and iii) the database service providing up-to-date information from biological databases. Each of these components run within Docker containers making the process of deploying the application simple.

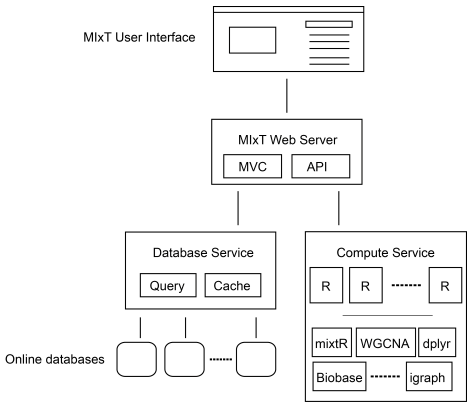


Figure 3.4: The architecture of the MIxT system. It consists of a web application, the hosting web server, a database service for retrieving metadata and a compute service for performing statistical analysis. Note that only the web application and the R package are specific to MIxT, the rest of the components can be reused in other applications.

The web application is hosted by a custom web server. This web server is responsible for dynamically generating the different views based on data from

6. Available online at `github.com/vdumeaux/mixtR`.

the statistical analyses and biological databases, and serve these to users. It also serves the different JavaScript visualization libraries and style sheets.

3.7 Related Work

OpenCPU is a system for embedded scientific computing and reproducible research.[22] Similar to the compute service in Kvik, it offers an HTTP API to the R programming language to provide an interface with statistical methods. It allows users to make function calls to any R package and retrieve the results in a wide variety of formats such as JSON or PDF. OpenCPU provides a JavaScript library for interfacing with R, as well as Docker containers for easy installation, and has been used to build multiple applications.[7]. The compute service in Kvik follows many of the design patterns in OpenCPU. Both systems interface with R packages using a hybrid state pattern over HTTP. Both systems provide the same interface to execute analyses and retrieve results. Because of the similarities in the interface to R in Kvik we provide packages for interfacing with our own R server or OpenCPU R servers.

Shiny is a web application framework for R[8] It allows developers to build web applications in R without having to have any knowledge about HTML, CSS, or Javascript. While it provides an easy alternative to build web applications on top of R, it cannot be used as a service in an application that implements the user-interface outside R.

Renjin is a JVM-based interpreter for the R programming language.[23] It allows developers to write applications in Java that interact directly with R code. This makes it possible to use Renjin to build a service for running statistical analyses on top of R. One serious drawback is that existing R packages must be re-built specifically for use in Renjin.

Cytoscape is an open source software platform for visualizing complex networks and integrating these with any type of attribute data.[24] Through a Cytoscape App, cyREST, it allows external network creation and analysis through a REST API[25], making it possible to use Cytoscape as a service. To bring the visualization and analysis capabilities to the web applications the creators of Cytoscape have developed Cytoscape.js[9], a JavaScript library to create interactive graph visualizations. Another alternative for biological data visualization in the web browser is BioJS It provides a community-driven on-

7. `opencpu.org/apps.html`.
8. `shiny.rstudio.com`.
9. `js.cytoscapejs.org`.

line repository with a wide range components for visualizing biological data contributed by the bioinformatics community.[7] BioJS builds on node.js[10] providing both server-side and client-side libraries. In MIxT we have opted to build the visualizations from scratch using sigma.js and d3 to have full control over the appearance and functionality of the visualizations.

3.7.1 Other Disciplines

We have also used the microservice architecture in an application where users can upload and explore air pollution data from Northern Norway.[26] In the project, air:bit, students from upper secondary schools in Norway collect air quality data from sensor kits that they have built and programmed. The web application lets the students upload data from their kits, and provides a graphical interface for them to explore data from their own, and other participating schools. The system consists of a web server frontend that retrieves air pollution data from a backend storage system to build interactive visualizations. It also integrates the data with other sources such as the Norwegian Institute for Air Research and the The Norwegian Meteorological Institute.

3.8 Evaluation

We evaluate the applications by investigating response times for a set of queries to each of the two supporting services.

To evaluate the database service we measure the query time for retrieving information about a specific gene with and without caching.[11] This illustrates how we can improve performance in an application by using a database service rather than accessing the database directly. We use a AWS EC2 *t2.micro*[12] instance to host and evaluate the database service. The results in Table 3.2 confirm a significant improvement in response time when the database service caches the results from the database lookups. In addition by serving the results out of cache we reduce the number of queries to the online database down to one.

We evaluate the compute service by running a benchmark consisting of two operations: first generate a set of 100 random numbers, then plot them and

10. `nodejs.org`.
11. More details online at `github.com/fjukstad/kvik/tree/master/db/benchmark`.
12. See `aws.amazon.com/ec2/instance-types` for more information about AWS EC2 instance types.

Table 3.2: Time to retrieve a gene summary for a single gene, comparing different number of concurrent requests.

	1	2	5	10	15
No cache	956ms	1123ms	1499ms	2147ms	2958ms
Cache	64ms	64ms	130ms	137ms	154ms

return the resulting visualization.[13] We use two *c4.large* instances on AWS EC2 running the Kvik compute service and OpenCPU base docker containers. The servers have caching disabled. Table 3.3 shows the time to complete the benchmark for different number of concurrent connections. We see that the compute service in Kvik performs better than the OpenCPU[14] alternative. We believe that speedup is because we keep a pool of R processes that handle requests. In OpenCPU a new R process is forked upon every request that results in any computation executed in R. Other requests such as retrieving previous results do not fork new R processes.

In summary our results show that the interface to the R programming language provides faster latencies, and that implementing a service for database lookups have clear benefits with regards to latency.

Table 3.3: Time to complete the benchmark with different number of concurrent connections.

	1	2	5	10	15
Kvik	274ms	278ms	352ms	374ms	390ms
OpenCPU	500ms	635ms	984ms	1876ms	2700ms

3.9 Discussion

There are different arguments for reusing and sharing microservices over libraries in bioinformatics applications, that would justify the cost of hosting an maintaining a set of distributed microservices. We argue that applications that require large computational or storage resources can benefit from the microservices approach because the applications can share the underlying compute infrastructure between multiple applications and users. This makes it possible to deploy an application on a lightweight system that uses a common service for computation and storage. In addition, benefits such as using different programming languages for a single application, and packaging a microservice as a software container, help to outweigh the operational burden related to

13. More details at `github.com/fjukstad/kvik/tree/master/r/benchmarks`.
14. Built using the *opencpu-server* Docker image.

using microservices to build applications.

We have reused the microservices for running statistical analyses and fetch biological metadata, and share these between applications. This makes it possible for multiple applications to use one or more powerful servers for hosting the services. In the case of statistical analyses we simply install the necessary R packages for each application on the compute service and run it as we would for one single application.

3.10 Conclusion

We have designed an approach for building data exploration applications in systems biology. Through an iterative approach we have identified a set of central components to these applications, and implemented them using a microservice architecture. Using this approach we have built a web application that integrates statistical analyses, interactive visualizations, and data from biological databases. While we have used our approach to build an application in systems biology, we believe that the microservice architecture can be used to build data exploration systems in other disciplines as well.

4

Deep Analysis Pipelines

In this chapter we discuss our approach to analyzing high-throughput genomic datasets through deep analysis pipelines, and its implementation in walrus.[**?**] We also evaluate the performance of walrus and show its usefulness in a precision medicine setting. While walrus was developed in this context we also show its usefulness in other areas, specifically for RNA-seq analyses.

4.1 Use Case and Motivation

For cancer, high throughput sequencing is an emerging technology to facilitate personalized diagnosis and treatment since it enables collecting high quality genomic data from patients at a low cost. Data collection is becoming cheaper, but the downstream computational analysis is still both a time consuming and thereby a costly part of the experiment. This is because of the computationally intensive methods needed to discover patterns in the data, in addition to the human resources to develop and use these methods in the analysis. Such analyses require deep analysis pipelines with a large number of steps that transform raw data into interpretable results.[6] These pipelines often consists of in-house or third party tools and scripts that each transform input files and produce some output. Although different tools exist, it is necessary to carefully explore different tools and parameters to choose the most efficient to apply for a dedicated question.[27] The complexity of the tools vary from toolkits such as the Genome Analysis Toolkit (GATK) to small custom *bash* or *R* scripts. In

addition some tools interface with databases whose versions and content will impact the overall result.[28]

When developing analysis pipelines for use in precision medicine it is necessary to track pipeline tool versions, their input parameters, and data. Both to thoroughly document what produced the end results, but also to compare results from different pipeline runs. Because of the iterative process of developing the analysis pipeline, it is necessary to use analysis tools that facilitate modifying pipeline steps and adding new ones with little developer effort.

We have previously analyzed DNA sequence data from a breast cancer patient's primary tumor and adjacent normal cells to identify the molecular signature of the patient's tumor and germline. When the patient later relapsed we analyzed sequence data from the patient's metastasis to provide an extensive comparison against the primary and to identify the molecular drivers of the patient's tumor.

We used Whole-genome sequencing (WGS) to sequence the primary tumor and adjacent normal cells at an average depth of 20, and Whole-exome sequencing (WES) at an average depth of 300. The biological samples were sequenced at the Genome Quebec Innovation Centre and we stored the raw datasets on our in-house server. From the analysis pipelines we generated reports with end results, such as detected somatic mutations, that was distributed to both the patient and the treating oncologists. These could be used to guide diagnosis and treatment, and give more detailed insight into both the primary and metastasis. When the patient relapsed we analyzed WES data using our own pipeline manager, `walrus`, to investigate the metastasis and compare it to the primary tumor.

For the initial WGS analysis we developed a pipeline to investigate somatic and germline mutations based on Broad Institute's best practices. We developed the analysis pipeline on our in-house compute server using a *bash* script version controlled with *git* to track changes as we developed the analysis pipeline. The pipeline consisted of tools including picard,[1] fastqc,[2] trimmomatic,[3] and the GATK.[4] While the analysis tools themselves provide the necessary functionality to give insights in the disease, ensuring that the analyses could be fully reproduced later left areas in need of improvement.

We chose a command-line script over more complex pipelining tools or work-

1. `broadinstitute.github.io/picard`
2. `bioinformatics.babraham.ac.uk/projects/fastqc`
3. `usadellab.org/cms/?page=trimmomatic`
4. `software.broadinstitute.com/gatk`

benches such as Galaxy[29] because of its fast setup time on our available compute infrastructure, and familiar interface. More complex systems could be beneficial in larger research groups with more resources to compute infrastructure maintenance, whereas command-line scripting languages require little infrastructure maintenance over normal use. In addition, while there are offsite solutions for executing scientific workflows, analyzing sensitive data often put hard restrictions on where the data can be stored and analyzed.

After we completed the first round of analyses we summarized our efforts and noted some lessons learned. First, datasets and databases should be version controlled and stored along with the pipeline description. In the analysis script we referenced to datasets and databases by their physical location on a storage system, but these were later moved without updating the pipeline description causing extra work. A solution would be to add the data to the same version control repository hosting the pipeline description. Second, the specific pipeline tools should also be kept available for later use. Since installing many bioinformatics tools require a long list of dependencies, it is beneficial to store the pipeline tools to reduce the time to start analyzing new data or re-run analyses. Another lesson is that it should be easy to add new tools to an existing pipeline and execution environment. This includes installing the specific tool and adding to an existing pipeline. Bundling tools within software containers, such as Docker, and hosting them on an online registry simplifies the tool installation process since the only requirement is the container runtime. Another lesson learned is that while bash scripts have their limitations, using a well-known format that closely resembles the normal command-line use clearly have its advantages. It is easy to understand what tools were used, their input parameters, and the data flow. However, from our experience when these analysis scripts grow too large they become too complex to modify and maintain. Last, while there are new and promising state-of-the art pipeline managers, many of these also require state-of-the-art computing infrastructure to run. This may not be the case for the current research and hospital environments.

The above problem areas are not just applicable to our research group, but common to other research and precision medicine projects as well. Especially when hospitals and research groups aim to apply personalized medicine efforts to guide therapeutic strategies and diagnosis, the analyses will have to be able to be easily reproducible later. We used the lessons learned to design and implement `walrus`, a command line tool for developing and running data analysis pipelines. It automatically orchestrates the execution of different tools, and tracks tool versions and parameters, as well as datasets through the analysis pipeline. It provides users a simple interface to inspect differences in pipeline runs, and retrieve previous analysis results and configurations. In the remainder of the paper we describe the design and implementation of `walrus`, its clinical

use, its performance, and how it relates to other pipeline managers.

4.2 walrus

walrus is a tool for developing and executing data analysis pipelines. It stores information about tool versions, tool parameters, input data, intermediate data, output data, as well as execution environments to simplify the process of reproducing data analyses. Users write descriptions of their analysis pipelines using a familiar syntax and walrus uses this description to orchestrate the execution of the pipeline. In walrus we package all tools in software containers to capture the details of the different execution environments. While we have used walrus to analyse high-throughput datasets in precision medicine, it is a general tool that can analyze any type of data, e.g. image datasets for machine learning. It has few dependencies and runs on on any platform that supports Docker containers. While other popular pipeline managers require the use of cluster computers or cloud environment, we focus on single compute nodes often found in clinical environments such as hospitals.

walrus is implemented as a command-line tool in the Go programming language. We use the popular software container implementation Docker[5] to provide reproducible execution environments, and interface with git together with git-lfs[6] to version control datasets and pipeline descriptions. By choosing Docker and git we have built a tool that easily integrates with current bioinformatic tools and workflows. It runs both natively or within its own Docker container to simplify its installation process.

With walrus we target pipeline developers that use command-line tools and scripting languages to build and run analysis pipelines. Users can use existing Docker containers from sources such as BioContainers,[30] or build containers with their own tools. We integrate with the current workflow using git to version control analysis scripts, and use git-lfs to expand to versioning of datasets as well. We have designed the pipeline description format resembles the command line syntax as much as possible. This is one of the major strengths of walrus. It uses a familiar syntax and format, and does not require the users to explicitly declare which files in the pipeline to version control.

5. docker.com
6. git-lfs.github.com

4.2.1 Pipeline Configuration

Users configure analysis pipelines by writing pipeline description files in a human readable format such as JavaScript Object Notation (JSON) or YAML Ain't Markup Language (YAML). A pipeline description contains a list of stages, each with inputs and outputs, along with optional information such as comments or configuration parameters such as caching rules for intermediate results. Listing 4.1 shows an example pipeline stage that uses MuTect[31] to detect somatic point mutations. Users can also specify the tool versions by selecting a specific Docker image, for example using MuTect version 1.1.7 as in Listing 4.1, line 3.

Users specify the flow of data in the pipeline within the pipeline description, as well as the dependencies between the steps. Since pipeline configurations can become complex, users can view their pipelines using an interactive web-based tool, or export their pipeline as a DOT file for visualization in tools such as Graphviz.[7]

Listing 4.1: Example pipeline stage for a tool that detects somatic point mutations. It reads a reference sequence file together with both tumor and normal sequences, and produces an output file with the detected mutations.

```
{
  "Name": "mutect",
  "Image": "fjukstad/mutect:1.1.7",
  "Cmd": [
    "--analysis_type","MuTect",
    "--reference_sequence","/walrus/input/reference.fasta",
    "--input_file:normal","/walrus/input/normal.bam",
    "--input_file:tumor","/walrus/input/tumor.bam",
    "-L","/walrus/input/targets.bed",
    "--out","/walrus/mutect/mutect-stats-txt",
    "--vcf","/walrus/mutect/mutect.vcf"
  ],
  "Inputs":[
     "input"
  ]
}
```

Users add data to an analysis pipeline by simply specifying the location of the input data in the pipeline description, and `walrus` automatically mounts it to the container running the analysis. The location of the input files can either be local or remote locations such as an FTP server. When the pipeline is completed, `walrus` will store all the input, intermediate and output data to a user-specified location.

7. graphviz.org

4.2.2 Pipeline Execution

When users have written a pipeline description for their analyses, they can use the command-line interface of `walrus` to run the analysis pipeline. `walrus` builds an execution plan from the pipeline description and runs it for the user. It uses the input and output fields of each pipeline stage to construct a directed acyclic graph (DAG) where each node is a pipeline stage and the links are input/output data to the stages. From this graph `walrus` can determine parallelizable stages and coordinate the execution of the pipeline.

In `walrus`, each pipeline stage is run in a separate container, and users can specify container versions in the pipeline description to specify the correct version of a tool. We treat a container as a single executable and users specify tool input arguments in the pipeline description file using standard command line syntax. `walrus` will automatically build or download the container images with the analysis tools, and start these with the user-defined input parameters and mount the appropriate input datasets. While the pipeline is running, `walrus` monitors running stages and schedules the execution of subsequent pipeline stages when their respective input data become available. We have designed `walrus` to execute an analysis pipeline on a single large server, but since the tools are run within containers, these can easily be orchestrated across a range of servers in future versions.

Users can select from containers pre-installed with bioinformatics tools, or build their own using a standard Dockerfile. Through software containers `walrus` can provide a reproducible execution environment for the pipeline, and containers provide simple execution on a wide range of software and hardware platforms. With initiatives such as BioContainers, researchers can make use of already existing containers without having to re-write their own. Data in each pipeline step is automatically mounted and made available within each Docker container. By simply relying on Docker `walrus` requires little software setup to run different bioinformatics tools.

While `walrus` executes a single pipeline on one physical server, it supports both data and tool parallelism, as well as any parallelization strategies within each tool, e.g. multi-threading. If users want to run the same analyses for a set of samples, or for example per chromosome, they can simply list the samples in the pipeline description and `walrus` will automatically run each sample through the pipeline in parallel. While we can parallelize the independent pipeline steps, the performance of an analysis pipeline relies on each of the independent tools and available compute power. This also applies to the scalability of the analysis pipeline.

Upon successful completion of a pipeline run, `walrus` will write a verbose

pipeline description file to the output directory. This file contains information on the runtime of each step, which steps were parallelized, and provenance related information to the output data from each step. Users can investigate this file to get a more detailed look on the completed pipeline. In addition to this output file `walrus` will return a unique version ID for the pipeline run, which later can be used to investigate a previous pipeline run.

4.2.3 Data Management

In `walrus` we provide an interface for users to track their analysis data through a version control system. This allows users to inspect data from previous pipeline runs without having to recompute all the data. `walrus` stores all intermediate and output data in an output directory specified by the user, which is version controlled automatically by `walrus` when new data is produced by the pipeline.

In `walrus` we interface with `git` to track any output file from the analysis pipeline. When users execute a pipeline, `walrus` will automatically add and commit output data t a git repository using `git-lfs`. Users typically use a single repository per pipeline, but can share the same repository to version multiple pipelines as well. With git-lfs, instead of writing large blobs to a repository it writes small pointer files that contains the hash of the original file, the size of the file, and the version of git-lfs used. The files themselves are stored separately which makes the size of the repository small and manageable with git. The main reason why we chose git and git-lfs for version control is that git is the de facto standard for versioning source code, and we want to include versioning of datasets without altering the typical development workflow.

Since we are working with potentially sensitive datasets `walrus` is targeted at users that use a local compute and storage servers. It is up to users to configure a remote tracker for their repositories, but we provide command-line functionality in `walrus` to run a git-lfs server that can store users' contents. They can use their default remotes, such as Github, for hosting source code but they must themselves provide the remote server to host their data.

4.2.4 Pipeline Reconfiguration and Re-execution

Reconfiguring a pipeline is common practice in precision medicine, e.g. if we later would like to change tool parameters or add additional steps to the analysis pipeline. To reconfigure an existing pipeline users make the applicable changes to the pipeline description and re-run it using `walrus`. `walrus` will

then recompute the necessary steps and return a version ID for the newly run pipeline. This ID can be used to compare pipeline runs, the changes made, and optionally restore the data and configuration from a previous run. Reconfiguring the pipeline to use updated tools or reference genomes will alter the pipeline configuration and force `walrus` to recompute the applicable pipeline stages.

The command-line interface of `walrus` provides functionality to restore results from a previous run, as well as printing information about a completed pipeline. To restore a previous pipeline run, users use the `restore` command line flag in `walrus` together with the version ID of the respective pipeline run. `walrus` will interface with git to restore the files to their state at the necessary point in time.

4.3 Evaluation

To evaluate the usefulness of `walrus` we demonstrate its use in a clinical setting, and the low computational time and storage overhead to support reproducible analyses.

4.3.1 Clinical Application

We have used `walrus` to analyze a whole-exome data from a sample in the McGill Genome Quebec [MGGQ] dataset (GSE58644)[12] to discover SNPs, genomic variants and somatic mutations. We interactively developed a pipeline description that follows the best-practices of The Broad Institute[8] and generated reports that summarized the findings to share the results. Figure 4.1 shows a screenshot from the web-based visualization in `walrus` of the pipeline.

From the analyses we discovered inherited germline mutations that are recognized to be among the top 50 mutations associated with an increased risk of familial breast cancer. We also discovered a germline deletion which has been associated with an increased risk of breast cancer. We also discovered mutations in a specific gene that might explain why specific drug had not been effective in the treatment of the primary tumor. From the profile of the primary tumor we discovered many somatic events (around 30 000) across the whole genome with about 1000 in coding regions, and 500 of these were coding for non-synonymous mutations. We did not see amplification or constituent activation of growth factors like HER2, EGFR or other players in breast cancer.

8. `software.broadinstitute.org/gatk/best-practices`

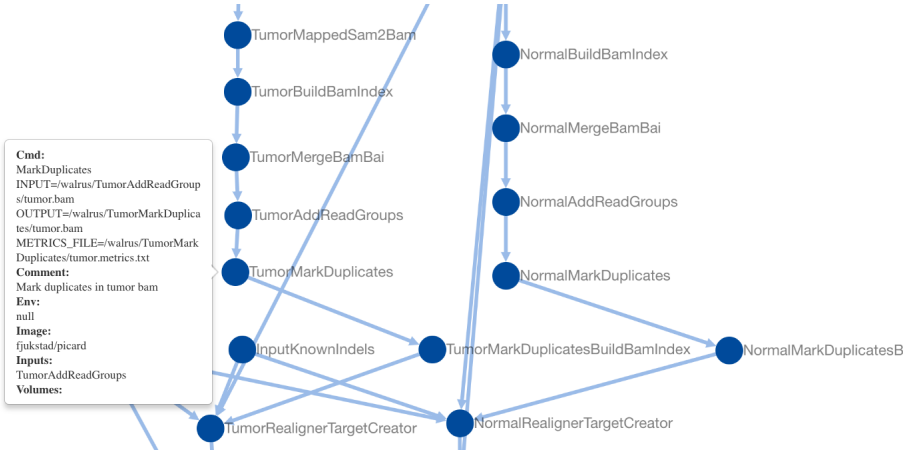


Figure 4.1: Screenshot of the web-based visualization in `walrus`. The user has zoomed in to inspect the pipeline step which marks duplicate reads in the tumor sequence data.

Because of the germline mutation, early recurrence, and lack of DNA events, we suspect that the patient's primary tumor was highly immunogenic. We have also identified several mutations and copy number changes in key driver genes. This includes a mutation in a gene that creates a premature stop codon, truncating one copy of the gene.

While we cannot share the results in details or the sensitive dataset, we have made the pipeline description available at `github.com/uit-bdps/walrus` along with other example pipelines.

4.3.2 Example Dataset

To demonstrate the performance of `walrus` and the ability to track and detect changes in an analysis pipeline, we have implemented one of the variant calling pipelines from [32] using tools from picard and the GATK. We show the storage and computational overhead of our approach, and the benefit of capturing the pipeline specification using a pipeline manager rather than a methods section in a paper. The pipeline description and code is available along with `walrus` at `github.com/uit-bdps/walrus`. Figure 4.2 shows a simple graphical representation of the pipeline.

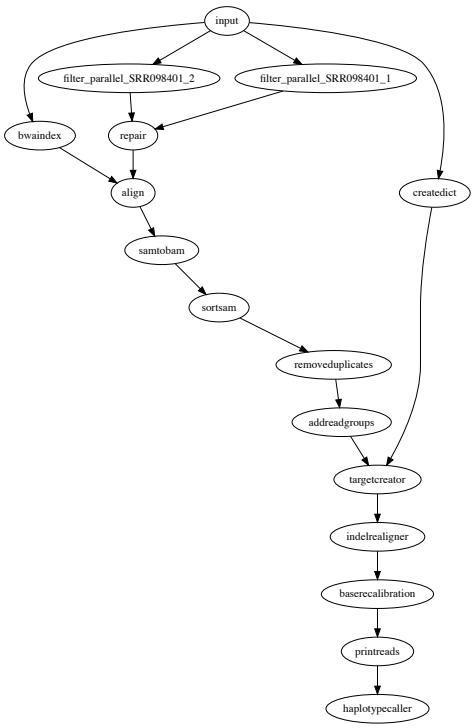


Figure 4.2: In addition to the web-based inteactive pipeline visualization, `walrus` can also generate DOT representations of pipelines. The figure shows the example variant calling pipeline.

4.3.3 Performance and Resource Usage

We first run the variant calling pipeline without any additional provenance tracking or storing of output or intermediate datasets. This is to get a baseline performance measurement for how long we expect the pipeline to run. We then run a second experiment to measure the overhead of versioning output and intermediate data. Then we introduce a parameter change in one of the pipeline steps which results in new intermediate and output datasets. Specifically we change the `-maxReadsForRealignment` parameter in the indel realigner step back to its default (See the online pipeline description for more details). This forces `walrus` to recompute the indel realigner step and any subsequent steps. We then use the `restore` flag in `walrus` to illustrate what the parameter change had on the pipeline output. To illustrate how `walrus` can restore old pipeline configurations and results, we restore the pipeline to the initial configuration and results. We show the computational overhead and storage usage of restoring a previous pipeline configuration.

Note that while the authors of [32] lists the actual commands of many of the pipeline steps, more than most publications, using the methods section of the paper to reproduce the analysis pipeline is still a difficult task. For example, the online version of [32] the parameters prefixed with two consecutive hypens (-) are converted to single em dashes (—). PDF versions of the paper lists the parameters correctly. In addition, the input filenames in the variant calling step do not correspond to any output files in previous steps, but because of their similarity to previous output files we assume that this is just a typo. These issues in addition to missing commands for e.g. the filtering step highlights the clear benefit of writing and reporting the analysis pipeline using a tool such as `walrus`.

Table 4.1 shows the runtime and storage use of the different experiments. In the second experiment we can see the added overhead of adding version control to the dataset. In total, an hour is added to the runtime and the data size is doubled. The doubling comes from git-lfs hard copying the data into a subdirectory of the `.git` folder in the repository. With git-lfs users can move all datasets to a remote server reducing the local storage requirements. In the third experiment we can see that only the downstream analyses from configuring the indel realignment parameter is executed. It generates 30GB of additional data, but the execution time is limited to the applicable stages. Restoring the pipeline to a previous configuration is almost instantaneous since the data is already available locally and git only has to modify the pointers to the correct files in the `.git` subdirectory.

Table 4.1: Runtime and storage use of a the typical workflow of developing a variant-calling pipeline with `walrus`.

Experiment	Task	Runtime	Storage Use
1	Run pipeline with default configuration	21 hours 50 minutes	235 GB
2	Run the default pipeline with version control of data	23 hours 9 minutes	470 GB
3	Re-run the pipeline with modified indel realignment parameter	13 hours	500 GB
4	Restoring pipeline back to the default configuration	< 1 second	500GB

4.4 Related Work

There are a wealth of pipeline specification formats and workflow managers available. Some are targeted at users with programming experience while others provide simple GUIs. Here we describe the most popular systems for building data analysis pipelines. While most provide viable options for genomic analyses, we have found most to complex to install and maintain in clinical settings. We discuss tools that use the common CWL pipeline specification and systems that provide versioning of data.

CWL is a specification for describing analysis workflows and tools.[3] A pipeline is written as a JSON or YAML file, or a mix of the two, and describes each step in detail, e.g. what tool to run, its input parameters, input data and output data. The pipeline descriptions are text files that can be version controlled and shared between projects. There are multiple implementations of CWL workflow platforms, e.g. the reference implementation cwl_runner,[9] Arvados,[33] Rabix,[34] Toil,[5] Galaxy,[29] and AWE.[35] It is no requirement to run tools within containers, but implementations can support it. There are few of these tools that support versioning of the data. Galaxy is an open web-based platform for reproducible analysis of large high-throughput datasets.[29] It is possible to

9. `github.com/common-workflow-language/cwltool`

run Galaxy on local compute clusters, in the cloud, or using the online Galaxy site[10] In Galaxy users set up an analysis pipeline using a web-based graphical interface, and it is also possible to export or import an existing workflow to an Extensible Markup Language (XML) file.[11] We chose not to use Galaxy because of missing command-line and scripting support, along with little support for running workflows with different configurations.[2] Rabix provides checksums of output data to verify it against the actual output from the pipeline. This is similar to the checksums found in the git-lfs pointer files, but they do not store the original files for later. Arvados stores the data in a distributed storage system, Keep, that provides both storage and versioning of data. We chose not to use CWL and its implementations because of its relaxed restrictions on having to use containers, its verbose pipeline descriptions, and the complex compute architecture required for some of the runners. We are however experimenting with an extension to `walrus` that translates pipeline descriptions written in `walrus` to CWL pipeline descriptions.

Pachyderm is a system for running big data analysis pipelines. It provides complete version control for data and leverages the container ecosystem to provide reproducible data processing.[12] Pachyderm consists of a file system (Pachyderm File System (PFS)) and a processing system (Pachyderm Processing System (PPS)). PFS is a file system with git-like semantics for storing data used in data analysis pipelines. Pachyderm ensures complete analysis reproducibility by providing version control for datasets in addition to the containerized execution environments. Both PFS and PPS is implemented on top of Kubernetes.[13] We believe that the approach in Pachyderm with version controlling datasets and containerizing each pipeline step is the correct approach to truly reproducible data analysis pipelines. The reason we did not use Kubernetes and Pachyderm was because our compute infrastructure did not support it. In addition we did not want to use a separate tool, PFS, for data versioning, we wanted to integrate it with the current pratice of using git for versioning.

As discussed in [36], recent projects propose to use containers for life science research. The BioContainers[30] and Bioboxes[37] projects addresses the challenge of installing bioinformatics data analysis tools by maintaining a repository of docker containers for commonly used data analysis tools. Docker containers are shown to have better than, or equal performance as VMs.[38] Both forms of virtualization techniques introduce overhead in I/O-intensive workloads, especially in VMs, but introduce negligible CPU and memory overhead. For genomics pipelines the overhead of Docker containers will be negligible since

10. Available at `usegalaxy.org`.
11. An alpha version of Galaxy with CWL support is available at `github.com/common-workflow-language/galaxy`.
12. `pachyderm.io`
13. `kubernetes.io`

these tend to be compute intensive and they typically run for several hours [38] Containers have also been proposed as a solution to improve experiment reproducibility, by ensuring that the data analysis tools are installed with the same responsibilities.[39]

4.5 Discussion

Precision medicine requires flexible analysis pipelines that allow researchers to explore different tools and parameters to analyze their data. While there are best practices to develop analysis pipelines for genomic datasets, e.g. to discover genomic variants, there is still no de-facto standard for sharing the detailed descriptions to simplify re-using and reproducing existing work. Pipelines typically need to be tailored to fit each project and patient, and different patients will typically elicit different molecular patterns that require individual investigation. While we could follow best practices to develop our pipeline we explored different tools and parameters before we arrived at the final analysis pipeline. For example, in our WES pipeline we ran several rounds of preprocessing (trimming reads and quality control) before we were sure that the data was ready for analysis. Having a pipeline system that could keep track of different intermediate datasets, along with the pipeline specification, simplifies the task of comparing the results from pipeline tools and input parameters. While we have developed one approach to version control genomic datasets in an analysis pipeline, we believe that there is still room for improvement.

While we provide one approach to version control datasets, there are still some drawbacks. git-lfs supports large files, but it still takes a significant amount of time to add large files to a repository. This makes the entire analysis pipeline slower, but we argue that having the files version controlled outweigh the runtime. In addition, there are only a few public gif-lfs hosting platforms for datasets larger than a few gigabytes, making it necessary to host these in house.

We aim to investigate the performance of running analysis pipelines with `walrus`, and the potential benefit of its built-in data parallelism. While our WES analysis pipeline successfully run steps in parallel for the tumor and adjacent normal tissue, we have not demonstrated the benefit of doing so. This includes benchmarking and analyzing the system requirements for doing precision medicine analyses. We are also planning on exploring parallelism strategies where we can split an input dataset into chromosomes and run some steps in parallel for each chromosome, before merging the data again.

4.6 Conclusions

We have designed and implemented `walrus`, a tool for developing reproducible data analysis pipelines for use in precision medicine. Precision medicine requires that analyses are run on hospital compute infrastructures and results are fully reproducible. By packaging analysis tools in software containers, and tracking both intermediate and output data, `walrus` provides the foundation for reproducible data analyses in the clinical setting. We have used `walrus` to analyze a patient's metastatic lesions and adjacent normal tissue to provide insights and recommendations for cancer treatment.

5

Conclusion

5.1 Lessons Learned

5.2 Broader Impact

5.3 Future Work

We intend to address few points we aim to address in future work, both in the MIxT web application as well as the supporting microservices. The first issue is to improve the user experience in the MIxT web application. Since it is executing many of the analyses on demand, the user interface may seem unresponsive. We are working on mechanisms that gives the user feedback when the computations are taking a long time, but also reducing analysis time by optimizing the underlying R package. The database service provides a sufficient interface for the MIxT web application. While we have developed the software packages for interfacing with more databases, these haven't been included in the database service yet. In future versions we aim to make the database service an interface for all our applications. We also aim to improve how we capture data provenance. We aim to provide database versions and meta-data about when a specific item was retrieved from the database. One large concern that we haven't addressed in this paper is security. In particular one security concern that we aim to address in Kvik is the restrictions on the execution of code in the compute service. We aim to address this in the

next version of the compute service, using methods such as AppArmor[1] that can restrict a program's resource access. In addition to code security we will address data access, specifically put constraints on who can access data from the compute service. We also aim to explore different alternatives for scaling up the compute service. Since we already interface with R we can use the Sparklyr[2] or SparkR[3] packages to run analyses on top of Spark.[**?**] Using Spark as an execution engine for data analyses will enable applications to explore even larger datasets.

1. `wiki.ubuntu.com/AppArmor`.
2. `spark.rstudio.com`.
3. `spark.apache.org/docs/latest/sparkr.html`.

A

Publications

Bibliography

[1] J. Goecks, A. Nekrutenko, and J. Taylor, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome biology*, vol. 11, no. 8, p. R86, 2010.

[2] O. Spjuth, E. Bongcam-Rudloff, G. C. Hernández, L. Forer, M. Giovacchini, R. V. Guimera, A. Kallio, E. Korpelainen, M. M. Kańduła, M. Krachunov *et al.*, "Experiences with workflows for automating data-intensive bioinformatics," *Biology direct*, vol. 10, no. 1, p. 43, 2015.

[3] P. Amstutz, R. Andeer, B. Chapman, J. Chilton, M. R. Crusoe, R. Valls Guimera, G. Carrasco Hernandez, S. Ivkovic, A. Kartashov, J. Kern *et al.*, "Common workflow language, draft 3," *figshare*, 2016.

[4] J. Köster and S. Rahmann, "Snakemake—a scalable bioinformatics workflow engine," *Bioinformatics*, vol. 28, no. 19, pp. 2520–2522, 2012.

[5] J. Vivian, A. A. Rao, F. A. Nothaft, C. Ketchum, J. Armstrong, A. Novak, J. Pfeil, J. Narkizian, A. D. Deran, A. Musselman-Brown *et al.*, "Toil enables reproducible, open source, big biomedical data analyses," *Nature Biotechnology*, vol. 35, no. 4, pp. 314–316, 2017.

[6] Y. Diao, A. Roy, and T. Bloom, "Building highly-optimized, low-latency pipelines for genomic data analysis." in *CIDR*, 2015.

[7] J. Gómez, L. J. García, G. A. Salazar, J. Villaveces, S. Gore, A. García, M. J. Martín, G. Launay, R. Alcántara, N. Del-Toro *et al.*, "Biojs: an open source javascript framework for biological data visualization," *Bioinformatics*, vol. 29, no. 8, pp. 1103–1104, 2013.

[8] B. Fjukstad, V. Dumeaux, K. S. Olsen, E. Lund, M. Hallett, and L. A. Bongo, "Building applications for interactive data exploration in systems biology," in *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*. ACM, 2017, pp. 556–561.

[9] B. Fjukstad, K. S. Olsen, M. Jareid, E. Lund, and L. A. Bongo, "Kvik: three-tier data exploration tools for flexible analysis of genomic data in epidemiological studies," *F1000Research*, vol. 4, 2015.

[10] K. S. Olsen, C. Fenton, L. Frøyland, M. Waaseth, R. H. Paulssen, and E. Lund, "Plasma fatty acid ratios affect blood gene expression profiles-a cross-sectional study of the norwegian women and cancer post-genome cohort," *PLoS One*, vol. 8, no. 6, p. e67270, 2013.

[11] V. Dumeaux, B. Fjukstad, H. E. Fjosne, J.-O. Frantzen, M. M. Holmen, E. Rodegerdts, E. Schlichting, A.-L. Børresen-Dale, L. A. Bongo, E. Lund *et al.*, "Interactions between the tumor and the blood systemic response of breast cancer patients," *PLoS Computational Biology*, vol. 13, no. 9, p. e1005680, 2017.

[12] A. Tofigh, M. Suderman, E. R. Paquet, J. Livingstone, N. Bertos, S. M. Saleh, H. Zhao, M. Souleimanova, S. Cory, R. Lesurf *et al.*, "The prognostic ease and difficulty of invasive breast carcinoma," *Cell reports*, vol. 9, no. 1, pp. 129–142, 2014.

[13] B. Fjukstad and L. A. Bongo, "A review of scalable bioinformatics pipelines," *Data Science and Engineering*, vol. 2, no. 3, pp. 245–251, 2017.

[14] Y. Kiselev, S. Andersen, C. Johannessen, B. Fjukstad, K. S. Olsen, H. Stenvold, S. Al-Saad, T. Donnem, E. Richardsen, R. M. Bremnes *et al.*, "Transcription factor pax6 as a novel prognostic factor and putative tumour suppressor in non-small cell lung cancer," *Scientific reports*, vol. 8, no. 1, p. 5059, 2018.

[15] J. D. Watson, F. H. Crick *et al.*, "Molecular structure of nucleic acids," *Nature*, vol. 171, no. 4356, pp. 737–738, 1953.

[16] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, R. J. Mural, G. G. Sutton, H. O. Smith, M. Yandell, C. A. Evans, R. A. Holt *et al.*, "The sequence of the human genome," *science*, vol. 291, no. 5507, pp. 1304–1351, 2001.

[17] I. H. G. S. Consortium *et al.*, "Initial sequencing and analysis of the human genome," *Nature*, vol. 409, no. 6822, p. 860, 2001.

[18] S. D. Kahn, "On the future of genomic data," *science*, vol. 331, no. 6018, pp. 728–729, 2011.

[19] N. R. Council *et al.*, *Toward precision medicine: building a knowledge network for biomedical research and a new taxonomy of disease*. National

Academies Press, 2011.

[20] I. F. Tannock and J. A. Hickman, "Limits to personalized cancer medicine," *N Engl J Med*, vol. 375, no. 13, pp. 1289–1294, 2016.

[21] A. Sboner, X. J. Mu, D. Greenbaum, R. K. Auerbach, and M. B. Gerstein, "The real cost of sequencing: higher than you think!" *Genome biology*, vol. 12, no. 8, p. 125, 2011.

[22] J. Ooms, "The opencpu system: Towards a universal interface for scientific computing through separation of concerns," *arXiv preprint arXiv:1406.4806*, 2014.

[23] A. Bertram, "Renjin: The new r interpreter built on the jvm," in *The R User Conference, useR! 2013 July 10-12 2013 University of Castilla-La Mancha, Albacete, Spain*, vol. 10, no. 30, 2013, p. 105.

[24] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker, "Cytoscape: a software environment for integrated models of biomolecular interaction networks," *Genome research*, vol. 13, no. 11, pp. 2498–2504, 2003.

[25] K. Ono, T. Muetze, G. Kolishovski, P. Shannon, and B. Demchak, "Cyrest: Turbocharging cytoscape access for external tools via a restful api," *F1000Research*, vol. 4, 2015.

[26] B. Fjukstad, N. Angelvik, M. W. Hauglann, J. S. Knutsen, M. Grønnesby, H. Gunhildrud, and L. A. Bongo, "Low-cost programmable air quality sensor kits in science education," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM, 2018, pp. 227–232.

[27] N. Servant, J. Roméjon, P. Gestraud, P. La Rosa, G. Lucotte, S. Lair, V. Bernard, B. Zeitouni, F. Coffin, G. Jules-Clément *et al.*, "Bioinformatics for precision medicine in oncology: principles and application to the shiva clinical trial," *Frontiers in genetics*, vol. 5, 2014.

[28] A. Sboner and O. Elemento, "A primer on precision medicine informatics," *Briefings in bioinformatics*, vol. 17, no. 1, pp. 145–153, 2015.

[29] J. Goecks, A. Nekrutenko, and J. Taylor, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome biology*, vol. 11, no. 8, p. R86, 2010.

[30] BioContainers, "Biocontainers," https://biocontainers.pro, 2017, [Online; Accesssed: 16.08.2017].

[31] K. Cibulskis, M. S. Lawrence, S. L. Carter, A. Sivachenko, D. Jaffe, C. Sougnez, S. Gabriel, M. Meyerson, E. S. Lander, and G. Getz, "Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples," *Nature biotechnology*, vol. 31, no. 3, pp. 213–219, 2013.

[32] A. Cornish and C. Guda, "A comparison of variant calling pipelines using genome in a bottle as a reference," *BioMed research international*, vol. 2015, 2015.

[33] Arvados, "Arvados | open source big data processing and bioinformatics," https://arvados.org, 2017, [Online; Accesssed: 16.08.2017].

[34] G. Kaushik, S. Ivkovic, J. Simonovic, N. Tijanic, B. Davis-Dusenbery, and D. Kural, "Rabix: an open-source workflow executor supporting recomputability and interoperability of workflow descriptions," in *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, vol. 22. NIH Public Access, 2016, p. 154.

[35] W. Tang, J. Wilkening, N. Desai, W. Gerlach, A. Wilke, and F. Meyer, "A scalable data analysis platform for metagenomics," in *Big Data, 2013 IEEE International Conference on*. IEEE, 2013, pp. 21–26.

[36] I. A. Raknes, B. Fjukstad, and L. Bongo, "nsroot: Minimalist process isolation tool implemented with linux namespaces," *Norsk Informatikkonferanse*, 2017.

[37] P. Belmann, J. Dröge, A. Bremges, A. C. McHardy, A. Sczyrba, and M. D. Barton, "Bioboxes: standardised containers for interchangeable bioinformatics software," *Gigascience*, vol. 4, no. 1, p. 47, 2015.

[38] P. Di Tommaso, E. Palumbo, M. Chatzou, P. Prieto, M. L. Heuer, and C. Notredame, "The impact of docker containers on the performance of genomic pipelines," *PeerJ*, vol. 3, p. e1273, 2015.

[39] C. Boettiger, "An introduction to docker for reproducible research," *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, 2015.