

Toward 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 altfor sjeldent 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 need for the development of new software tools and packages to enable novel insights to the underlying biological mechanisms, in the development, and progression, of diseases such as cancer.

The heterogeneity of biological datasets require researchers to tailor the exploration and analyses using a range of different tools and systems. However, despite the need for their integration, few of them 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 necessary to record these correctly. The lack of such details complicates the reproducing of original results and reuse of the analyses on new datasets. This increases time for analysis and leaves unrealized potential for scientific insights.

This dissertation argues that we can develop unified systems for reproducible exploration and analysis of high-throughput biological datasets. We propose an approach, Small Modular Entities (SMEs), for developing data analysis pipelines and data exploration applications in cancer research. 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 needed to reproduce the analyses.

Through this approach we have developed different applications for analyzing high-throughput DNA sequencing datasets, and exploring gene expression data integrated with questionnaires, registry, and online databases. The evaluation shows how we effectively capture provenance in analysis pipelines and exploration applications. This simplifies reproducibility through the sharing of methods, tools, and data, and is fundamental to science.

Acknowledgements

First I would like to thank my advisor, Professor Lars Ailo Bongo for his relentless support and encouragement during my time as a PhD student. He has indeed shown me what tough love is, and I think that I am grateful for that.

I would like to thank my co-advisors Professor Eiliv Lund and Associate Professor Karia Standahl Olsen for their wonderful ideas and warm welcome into a research field that was not my own.

I would like to extend my gratitude to Professor Michael Hallett and Vanessa Dumeaux for their hospitality when I visited their lab in Montreal in 2016. I do not think this thesis would have been as interesting, without the projects I was fortunate enough to be a part of. Thank you!

I would like to thank my long time office wife Einar Holsbø, Morten Grønnesby, Nina Angelvik, and the BDPS lab at UiT.

Thank you to past or current students at UiT: Jan-Ove 'Kuken' Karlberg, Vegard Sandengen, Helge Hoff, Magnus Stenhaug, Erlend Graff, Kristian Elsebø, Martin Ernstsen, Amund Jensen, Michael Kampffmeyer, to name a few. You have all contributed to why I have been at the University for almost a decade.

I would like to thank my colleagues at the Department of Computer Science, especially the technical staff, led by Maria Wulff Hauglann, down the hall from my old office. Also, the administrative staff at our School lab for their wonderful collaboration.

Thank you to the PhD students at Nordlandssykehuset in Bodø who have been my closest colleagues during the final push of my PhD.

I would like to thank my mom and dad, and my younger brother for their ever-present support.

Finally, Ane Sætrum for her continuous love and support, and her *endurance* through all of my big or small projects.

Contents

Abstract	iii
Acknowledgements	v
List of Figures	xi
List of Tables	xiii
List of Abbreviations	xv
1 Introduction	1
1.1 Problems with Data Analysis and Exploration in Bioinformatics	4
1.2 Small Modular Entities (SMEs)	5
1.2.1 Data Management and Analysis	6
1.2.2 Interactive Data Exploration Applications	7
1.2.3 Deep Analysis Pipelines	8
1.2.4 Similarity	8
1.3 Applications Developed with SMEs	9
1.3.1 Data Management and Analysis	9
1.3.2 Interactive Data Exploration Applications	9
1.3.3 Deep Analysis Pipelines	10
1.4 Summary of Results	11
1.5 List of papers	11
1.6 Dissertation Plan	15
2 Modern Biological Data Management and Analysis	17
2.1 High-Throughput Datasets for Research and Clinical Use	18
2.2 Norwegian Women and Cancer (NOWAC)	19
2.2.1 Data Management and Analysis	19
2.3 Enabling Reproducible Research	20
2.3.1 The nowac Package	22
2.4 Standardized Data Analysis	24
2.4.1 Pipeline	24
2.5 Best Practices	24

2.6	Discussion	27
2.7	Conclusion	28
3	Interactive Data Exploration Applications	29
3.1	Motivating Examples	31
3.1.1	High and Low Plasma Ratios of Essential Fatty Acids	31
3.1.2	Tumor-Blood Interactions in Breast Cancer Patients	31
3.2	Requirements	32
3.3	Kvik Pathways	32
3.3.1	Analysis Tasks	33
3.3.2	Architecture	33
3.3.3	Implementation	34
3.3.4	Use Case: Analysis of Renin-Antiotensin Pathway	36
3.4	Design Principles	36
3.5	Kvik	36
3.5.1	Microservices	37
3.6	MIXT	39
3.6.1	Analysis Tasks	39
3.6.2	Architecture	40
3.6.3	Implementation	40
3.6.4	Evaluation	41
3.6.5	Tumor epithelium-stroma interactions in breast cancer	43
3.7	Related Work	43
3.7.1	air:bit	44
3.8	Discussion	45
3.9	Future Work	45
3.9.1	MIXT	45
3.10	Conclusion	46
4	Deep Analysis Pipelines	47
4.1	Use Case and Motivation	47
4.1.1	Initial Data Analysis Pipeline	48
4.2	walrus	50
4.2.1	Pipeline Configuration	51
4.2.2	Pipeline Execution	52
4.2.3	Data Management	53
4.2.4	Pipeline Reconfiguration and Re-execution	54
4.3	Results	54
4.3.1	Clinical Application	54
4.3.2	Example Dataset	55
4.3.3	Performance and Resource Usage	57
4.4	Related Work	58
4.5	Discussion	60
4.6	Conclusions	61

5 Conclusion	63
5.1 Lessons Learned	64
5.2 Future Work	65
Bibliography	69

List of Figures

1.1	The applications and their underlying systems discussed in this thesis.	5
1.2	The SME approach in different systems.	6
2.1	A screenshot of the user interface of R Studio.	23
2.2	Standardized data processing pipeline	25
2.3	A screenshot of the web-interface of Pippeline.	26
3.1	Screenshot of the renin-angiotensin pathway in Kvik Pathways	34
3.2	The three-tiered architecture of Kvik Pathways.	35
3.3	MIxT module overview page.	41
3.4	The architecture of the MIxT system.	42
4.1	Screenshot of the web-based visualization in <i>walrus</i>	55
4.2	DOT representations of a pipeline in <i>walrus</i>	56

List of Tables

3.1	The REST interface to the Data Engine, for example, use /genes/ to retrieve all available genes in our dataset.	34
3.2	Time to retrieve a gene summary for a single gene, comparing different number of concurrent requests.	42
3.3	Time to complete the benchmark with different number of concurrent connections.	43
4.1	Runtime and storage use of the example variant-calling pipeline developed with <code>walrus</code>	58

List of Abbreviations

CLI Command-line Interface

CRAN Comprehensive R Archive Network

CSV Comma-separated values

CWL Common Workflow Language

DAG directed acyclic graph

DNA Deoxyribonucleic acid

GATK Genome Analysis Toolkit

GB Gigabyte

GPU graphical processing unit

GUI Graphical User Interface

HTS High-throughput Sequencing

IDE integrated development environment

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

VM Virtual Machine

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 decreasing costs of data collection. This brings opportunities for gaining novel insights into the underlying biological mechanisms in the development and progression of diseases such as cancer, possibly leading to the development of new 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 detailed information about the analyses. This leads to unnecessary difficulties when reusing known methods, and reproducing the analyses, which in turn leads to a longer analysis process and therefore unrealized potential for scientific insights. For clinicians, inaccurate results from improperly developed analyses can lead to negative consequences for patient care.^[1]

We have identified four main challenges for application developers to undertake when building systems for analyzing and exploring biological datasets in research and the clinic. 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 challenge is managing datasets and analysis code for use by data exploration applications and data analysis pipelines. This includes storing all information that is necessary to

a data analyst when he or she is interpreting the data, as well as any analysis code that can be used to analyze the data. The second challenge is to develop data exploration applications that provide sufficient information to fully document every step that went into the analyses up to an end result. This includes reporting input parameters, tool versions, database versions, and dataset versions. The third challenge is developing applications that require the integration of disparate systems. These are often developed using different programming languages and provide different functionality, e.g., the combination of a web-based visualization with a graphical processing unit (GPU) accelerated statistical method, or the integration of a remote biological database. The final challenge is to develop applications and systems so that they can be easily shared and reused across research institutions.

As a result, there is a wealth of specialized approaches and systems to manage and analyze modern biological data. Systems such as Galaxy[2] provide simple Graphical User Interfaces (GUIs) for setting up and running 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.[3] With R and its popular package repository Bioconductor[4], researchers can select from a wide range of packages to tailor their analyses. These provide specialized analysis environments, but makes it necessary for the analyst to manually record information about data, tools, and tool versions. Systems such as Pachyderm[5] or the Common Workflow Language (CWL)[6] 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 for institutions without dedicated technical staff. Shiny[7] and OpenCPU[8] provide frameworks for application developers to build systems to interactively explore results from statistical analyses. These are useful for building 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 the development of reproducible data analysis and exploration systems for high-throughput biological data, through the integration of disparate systems and data sources. 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 Small Modular Entitys (SMEs), argues that applications for analyzing and exploring biological datasets should be modeled as a composition of individual systems and tools. We believe that the

Unix philosophy to "Do one thing and do it well"^[9] appropriately summarizes many existing tools in bioinformatics, and we should aim to build applications as compositions of these tools. Our SME approach resembles the traditional Unix-like pipelines, in combination with the service-oriented architecture^[10] or the microservice architectural style now popularized by web-scale distributed systems.^[11]

The approach has several key advantages when implementing systems to analyze and explore biological data:

- It enables and simplifies the development of applications that integrate disparate tools.
- It enables reproducible research by packaging applications and tools within containerized environments.
- With well-defined interfaces it is a simple task to add new components to a system, or modify existing ones.
- Through software container technology it becomes a simple task to deploy and scale up such applications.

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 extrapolated 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 often used for 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, there is a move towards using more sophisticated approaches for analyzing biological datasets using workflow and pipeline mangers such as Snakemake[12], and the different implementations of the CWL[13] such as Galaxy[2] and Toil[14]. These simplify setting up and executing the analysis pipeline. However, these tools still have their limitations, such as maintenance and tool updates. Other programming environments and scripting languages such as Python or R both provide a wide variety of software packages to read and process biological datasets. Especially the package repository Bioconductor[4] provides a long list of well-maintained software packages. Both these languages require the researchers to set up their own analyses, but can be tailored to fit their data precisely. For visually exploring biological data there are a range of tools, such as Cytoscape[15] and Circos[16], that support importing an already-analyzed dataset to visualize and browse the data. One problem with these are 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 full workflow for exploring or analyze biological data will have to combine multiple tools. The tools provide different interfaces and processing data often require data wrangling between the tools.
2. **Decoupling:** Data exploration tools are often decoupled from the statistical analyses. This often makes it a difficult to document and retrace the analyses through the full workflow.
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

		Data management and analysis	Interactive exploration	Deep analysis pipelines
Application	Pipeline	Kvik Pathways	MIxT	Clinical Sequencing Analysis
Underlying System	NOWAC R Package	Kvik		walrus
	Chapter 2	Chapter 3		Chapter 4

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

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 approach for unifying reproducible data analysis and exploration systems would reduce the time-to-interpretation of biological datasets significantly.

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.[17]. 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.

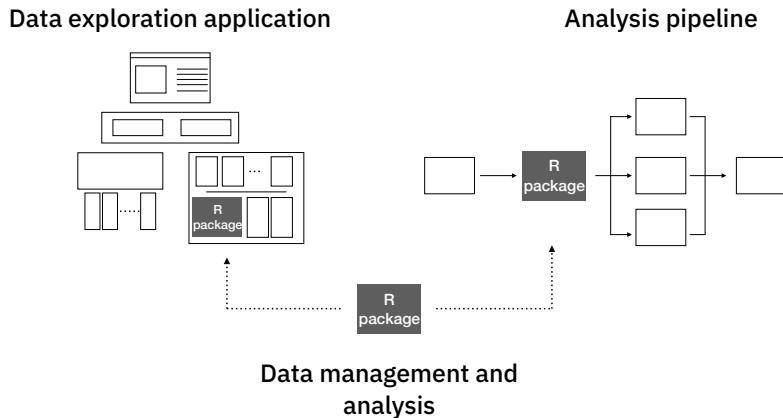


Figure 1.2: An illustration of how we envision the SME approach in data management systems, data exploration applications and analysis pipelines. In this example we reuse an R package for all use cases.

We discuss how the approach is suitable for different use cases before highlighting why it is suitable for all of them. Figure 1.2 shows the three different use cases and one such SME. We can use it in data exploration applications, analysis pipelines, and for building data management systems.

1.2.1 Data Management and Analysis

Modern epidemiological studies integrate traditional questionnaire data with information from public registries and biological datasets. These often span multiple biological levels, i.e., different data types and collection sites. While traditional survey based 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[4] 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 in in-house databases or shared file systems with specialized permissions. Together the packages and datasets form building blocks that researchers can develop their analyses on top of. They can

compose their analyses using packages that fit their specific needs. The analysis code in the NOWAC study may constitute such a 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 user-friendly preprocessing pipelining tool named Pippline.

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 use version control for both code and the data, making it possible to track changes over time as the research study evolves. Pippline is a web-based interface for running the standardized preprocessing steps before analyzing gene expression datasets in the NOWAC cohort.

1.2.2 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[18] now provide developers with tools to develop web applications for exploring biological datasets. It is apparent that these types of systems also consist of multiple smaller components that together can be orchestrated into a single application. These applications typically include 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 facilitate the integration 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.[19] *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[20] for exploring and comparing transcriptional profiles from blood and tumor samples in breast cancer patients, in addition to applications for exploring biological pathways[19].

1.2.3 Deep Analysis Pipelines

Analysis of high-throughput sequencing datasets requires deep analysis pipelines with many steps that transform raw data into interpretable results.[21] There are many tools available that 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 developers 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, we develop an analysis pipeline 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.

1.2.4 Similarity

The above approaches for building data analysis and exploration applications share the same design principles. In all areas we decompose the system, into small modular entities, 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 track the configuration of the containers and their orchestration using software versioning systems, and provide the necessary information to set up the system and reproduce their results. We believe that the SME approach is applicable to every step in the long process from raw data collection to interpretable results, and that it makes this process more transparent.

1.3 Applications Developed with SMEs

In this section we outline the different systems we have built using SMEs. We detail how we implemented SME in the NOWAC package, `walrus`, and `Kvik`, and show applications that use these.

1.3.1 Data Management and Analysis

To standardize the preprocessing of biological datasets in the NOWAC study. With the NOWAC package we could implement a preprocessing pipeline on top of it that used its datasets and utility functions to generate analysis-ready datasets for the researchers. This preprocessing pipeline called Pippeline was developed as a web application which allows the data managers in our study to generate datasets for researchers. The pipeline performs all necessary steps before researchers can perform their specialized analyses.

1.3.2 Interactive Data Exploration Applications

The first interactive data exploration application that we built was Kvik Pathways. It allows users to explore gene expression data from the NOWAC cohort in the context of interactive pathway maps.[19] 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.[22]

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 that we developed. Both of these were designed and implemented

in Kvik, which could then be used and shared later.

The second application that we built was 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.[23] 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 that we developed was a simple re-deployment of the `MIXT` web application with a new dataset. In this application that we simply replaced the R package with a new package that interfaced with different data. All the other components are reused. It demonstrates the flexibility of the approach.

1.3.3 Deep Analysis Pipelines

The first system that we built on top of `walrus` was a pipeline to analyze a patient's primary tumor and adjacent normal tissue, including subsequent metastatic lesions.[24] We packaged the necessary tools for the analyses into software containers and wrote a pipeline description with all the necessary data processing steps. Some 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. Like 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.

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

1.4 Summary of Results

We show the viability of our approach through real-world applications in systems epidemiology and precision medicine. Through our `nowac` package and `Pipeline`, we demonstrate its usefulness for enabling reproducible analyses of biological datasets in a complex epidemiological study. 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 from a sample in the McGill Genome Quebec [MGGQ] dataset (GSE58644)[25] 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 the 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	[19] 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,” <i>F1000Research</i> , 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	I 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	[20] 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 <i>Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics</i> . 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	I designed, implemented, and deployed the MIXT web application. Contributed to the writing of the manuscript.
Publication date	28 September 2017.
Publication venue	PLoS Computational Biology
Citation	[23] 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 <i>et al.</i> , “Interactions between the tumor and the blood systemic response of breast cancer patients,” <i>PLoS Computational Biology</i> , 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	I performed the literature review and wrote the manuscript.
Publication date	23 October 2017
Publication venue	Data Science and Engineering
Citation	[26] B. Fjukstad and L. A. Bongo, “A review of scalable bioinformatics pipelines,” <i>Data Science and Engineering</i> , 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	I contributed to the writing of the manuscript, specifically to the literature review and related works.
Publication date	26 November 2017
Publication venue	Norsk Informatikkonferanse 2017.
Citation	[27] I. A. Raknes, B. Fjukstad, and L. Bongo, “nsroot: Minimalist process isolation tool implemented with linux namespaces,” <i>Norsk Informatikkonferanse</i> , 2017

Paper 6

Title	Reproducible Data Analysis Pipelines for Precision Medicine
Authors	Bjørn Fjukstad , Vanessa Dumeaux, Michael Hallett, Lars Ailo Bongo
Description	This paper outlines how we used the SMEs approach to build walrus.
Contribution	I designed, implemented, and performed the evaluation of walrus. I also wrote the manuscript.
Publication	Submitted for publication. Preprint available online at biorxiv.org/content/early/2018/06/25/354811
Citation	[24] B. Fjukstad, V. Dumeaux, M. Hallett, and L. A. Bongo, “Reproducible data analysis pipelines for precision medicine,” https://www.biorxiv.org/content/early/2018/06/25/354811 , 2018

In addition to the above papers I have also contributed to the following papers during the project:

- 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

- 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

These are not included in the thesis but they demonstrate other usage examples of our approach.

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 Management and Analysis

From the discovery of the DNA structure by Watson and Crick in 1953[30] to the sequencing of the human genome in 2001 [31, 32] and the massively parallel sequencing platforms in the later years[33], 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.[34] 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 addition to faster data generation, the new datasets are also of higher quality.

Ensuring reproducibility through sharing of analysis code and datasets is necessary to advance science.[35] From the many obstacles to replicate results from the most influential papers in cancer research[36], it is apparent that it is important to thoroughly document the entire workflow from data collection to interpretable results. This requires implementing best practices for data storage and processing. Such best practices are also necessary for large and complex projects where data collection, analysis, and interpretation may span decades, and therefore be done in several iterations.

In this chapter we describe our efforts to establish an approach for reproducible analysis of biological data in a complex epidemiological study. We first give a

short introduction to high-throughput datasets, before describing the NOWAC study we use as a motivating example. We describe the previous practice for data management and analysis, and propose a new approach to achieve reproducible analyses. Continuing, we show that our approach to manage research data and code can be used to develop a standardized data analysis pipeline. Further we provide best practices for data analysis and management.

2.1 High-Throughput Datasets for Research and Clinical Use

It is important to have an understanding of the different high-throughput technologies that are now widely used to study complex diseases such as cancer. 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, microarrays have been used 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 many target genes, and by profiling these we can study which genes are active in the biological sample. Microarray datasets are in the size 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.[37] 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.[38] 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)

In this thesis we have used data from the NOWAC study extensively. The NOWAC study is a prospective population-based cohort that tracks 34% (170.000) of all Norwegian women born between 1943–57.[17] The data collection started in NOWAC in 1991 with surveys 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. The datasets are also integrated with data from The Norwegian Cancer Registry, and The Cause of Death Registry in Statistics Norway. In addition to the questionnaire data, the study includes blood samples from 50.000 women, as well as more than 300 biopsies. From the biological samples the first gene expression dataset was generated in 2009, and the study now also feature miRNA, methylation, metabolomics, and RNA-seq datasets.

The data in the NOWAC cohort allows for a number of different study designs. While it is a prospective cohort study, we can also draw a case-control study from the cohort, or a cross-section of the cohort. From the NOWAC cohort there has been published a number of research papers that investigate the questionnaire data together with the gene expression datasets.[22, 39] We have also used the gene expression datasets to explore gene expression signals in blood and interactions between the tumor and the blood systemic response of breast cancer patients.[40, 23]. Some analyses have resulted in patents[41] and commercialization efforts. While many interesting patterns and results have been studied, there are still many unexplored areas in the available datasets.

In the NOWAC study we are a traditional group of researchers, PhD and Post-Doc students, and administrative and technical staff. Researchers have backgrounds from statistics, medicine, or epidemiology, and now also computer science. The administrative and technical staff is responsible for managing the data, both data collection and data delivery to researchers.

2.2.1 Data Management and Analysis

Surveys are the traditional data collection method in epidemiology. But today, questionnaire responses are increasingly integrated with molecular data. However, surveys are still important for designing a study that can answer

particular research questions. In this section we describe how such integrated data analysis was done in NOWAC before we developed our approach. We believe many studies have, or are still, analyzing epidemiological data using a similar practice.

In the NOWAC study we have stored the raw survey and registry data in an in-house database backed up to an independent storage node. Previously, researchers had to apply to get data exported from the database by an engineer. This was typically done through SAS scripts that did some preprocessing, e.g. selecting applicable variables or samples, before the data was sent to researchers as SAS data files. The downstream analysis was typically done in SAS. Researchers used e-mail to communicate and send data analysis scripts, so there was not a central hub with all the scripts and data.

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 to our scientific staff, which are then processed into a standardized format. Since the NOWAC study is a prospective cohort, a percentage of the 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 processed and analyzed outside our research institution. The received raw datasets were then stored on a local server and made available to researchers on demand. Because of the complexity of the biological datasets, many of these require extensive pre-processing before they are analysis-ready.

2.3 Enabling Reproducible Research

To enable reproducible research in the NOWAC study we have developed a system for managing and documenting the available datasets, a standardized data preprocessing and preparation system, and a set of best practices for data analysis and management. We designed our management and analysis system as a SME that we could later use in the Pipeline system. Through experiences from transcriptomics data analysis we first identified issues with our previous practice:

1. It was difficult to keep track of the available datasets, and to determine how these had been processed. We had no standard data storage platform or structure, and there are limited reports for exported datasets used in different research projects.

2. There was no standard approach to preprocess and initiate data analysis. This was because the different datasets were analyzed by different researchers, and there was little practice for sharing reusable code between projects.
3. It became difficult to reproduce the results reported in our published research manuscripts. This was because we did not have any standardized preprocessing, shared the same analysis tools, or thoroughly documented the full analysis process.

To solve these issues and enable reproducible research in the NOWAC study, we had to develop a system for managing the data, code, and our proposed best practices for analyzing the data. We started with identifying a set of requirements for a system to manage and document the different datasets:

- It should provide users with a single interface to access the datasets, their respective documentation, and utility functions to access and analyze the data.
- It should provide version history for the data and analysis code.
- The system should provide reproducible data analysis reports¹ for any dataset that has been modified in any way.
- It should be portable and reusable by other systems or applications.

To satisfy the above requirements we developed the `nowac` R package, a software package in the R programming language that provides access to all data, documentation, and utility functions. Since it is a requirement that it should be reusable we could then implement a data preparation system, Pipeline, ontop of this R package. We identified a set of requirements for this data preprocessing and preparation system as well:

- The data preprocessing and preparation system should provide users with an interactive point-and-click interface to generate analysis-ready datasets from the NOWAC study.
- It should use the `nowac` R package to retrieve datasets.
- It should provide users with a list of possible options for filtering, normalization, and other options required to preprocess a microarray dataset.

¹. Such as an R Markdown file which, when executed, generates the output data and optional documentation including plots, tables etc.

- It should generate a reproducible report along with any exported dataset.

Finally, we developed a set of best practices for data analysis in our study. In the rest of the section we detail how we built the nowac package, the Pipeline, and the best practices for data analysis.

2.3.1 The nowac Package

The nowac R package is our solution for storing, documenting, and providing analysis functions to process the datasets in the NOWAC study. We use git to version control the analysis code and datasets, and store the repository on a self-hosted git server. We bundle together all datasets in the nowac package. This includes both questionnaire, registry, and gene expression datasets. Because none of these are particularly large (no single dataset being more than tens of GBs) we are able to distribute them with our R package. Some 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 and 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 clarity, we decorate the scripts with specially formatted comments that can be used with knitr[42] to generate reproducible data analysis reports. These highlight the transformation of the data from raw to clean, with information such as removed samples or data normalization methods.

We have documented every dataset in R package. The documentation includes information 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 the documentation is used to generate interactive help pages which they can browse in R, either through a command line or through an integrated development environment (IDE) such as RStudio. We can also export this documentation to a range of different formats, and researchers can also view them in the R interface. Figure 2.1 shows the user interface of RStudio where the user has opened the documentation page for one of the gene expression dataset.

In the NOWAC package we also 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.

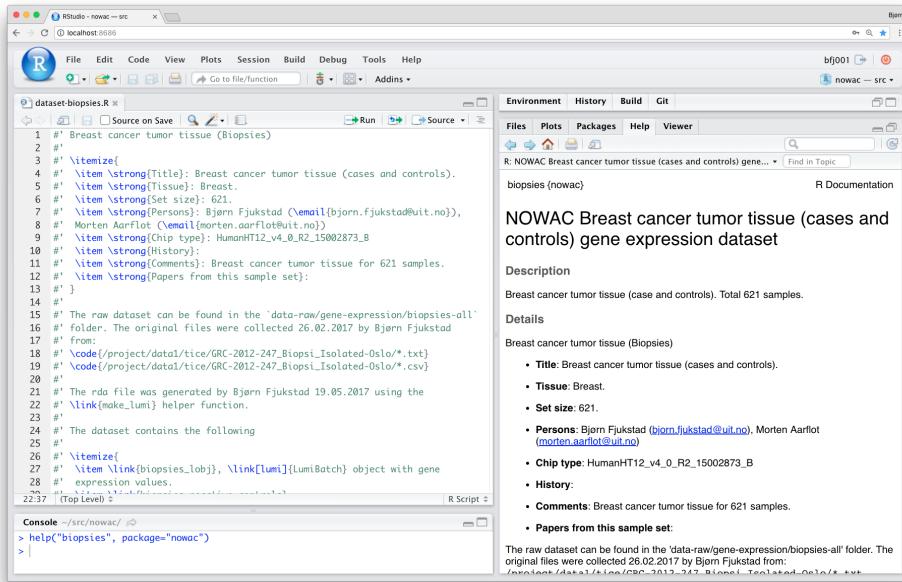


Figure 2.1: A screenshot of the user interface of R Studio viewing the documentation help page for the "Biopsies" dataset in the NOWAC study. The right-hand panel shows the documentation generated by the code in the top left panel. The bottom left panel shows the R command that brought up the help page.

We use a single repository for the R package, but have opted to use git submodules for datasets in the R package. This allows us to separate the access to the datasets, and the documentation and analysis code. Everyone with access to the repository can view the documentation and analysis code, but only superusers have access to the data. 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[43]. Submodules allow us to keep the main repository size down while still versioning the data. Other alternatives such as git-raw[44], git-annex[45] git-lfs[46] 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, or suit our preferred usage. Everyone with access to the repository can view the documentation and analysis code, but only technical staff have access to the data.

2.4 Standardized Data Analysis

Analyzing the biological data in the NOWAC study consists of four major parts as shown on Figure 2.2. First, as explained above the raw datasets are added to the nowac R package and documented thoroughly by a data manager. Second, we manually examine the biological datasets to detect outliers. We add information about outliers to the nowac R package along with reports that describe why an observation is marked as an outlier. Third, the data manager generates an analysis-ready dataset for a research project using the interactive Pippeline tool. This dataset is preprocessed, and integrated with questionnaire and registry datasets. Fourth, researchers analyze the dataset with their tools of choice, but following using our best practices for data analysis.

2.4.1 Pippeline

We have developed our preprocessing pipeline for gene expression data as a point-and-click web application called Pippeline. The web application is stand-alone and does not require the users to use any command-line tools or have any programming knowledge. Pippeline generates an analysis-ready dataset by integrating biological datasets together with questionnaire and registry data, all found in our nowac package. It uses pre-discovered outliers to exclude samples, and presents the user with a list of possible processing options. It exports the analysis-ready R data files together with a reproducible data analysis report, an R script, that describes all processing steps. Figure 2.3 shows the filtering step in Pippeline where users define at what level they wish to exclude gene expression probes in the dataset.

The web application is implemented in R using the Shiny framework. It uses the nowac R package to retrieve all datasets.

2.5 Best Practices

From our experiences we have developed a set of best practices for data analysis. These apply both to researchers, and developers and the technical staff managing the data in a research study:

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.

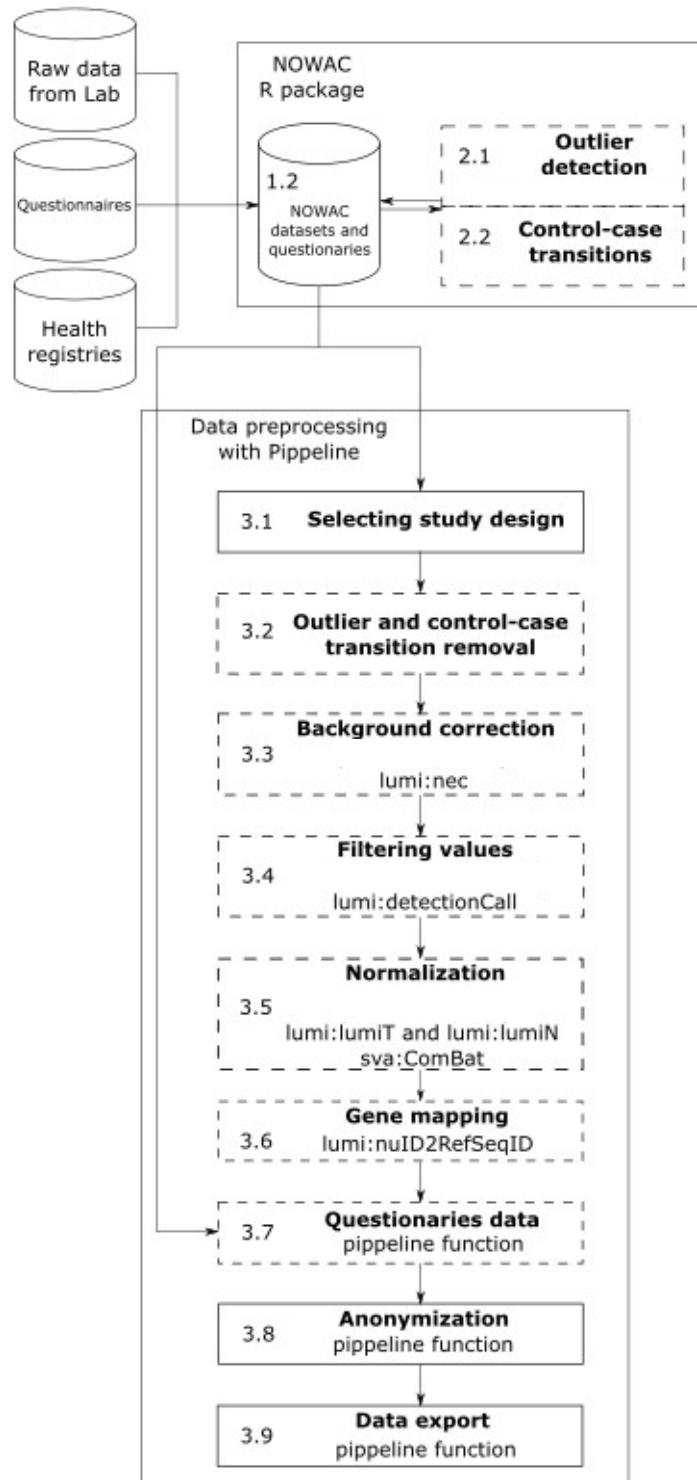


Figure 2.2: The standardized data processing pipeline for gene expression data analysis in the NOWAC study. Steps with a dashed line are optional, while steps marked with a solid line are mandatory.

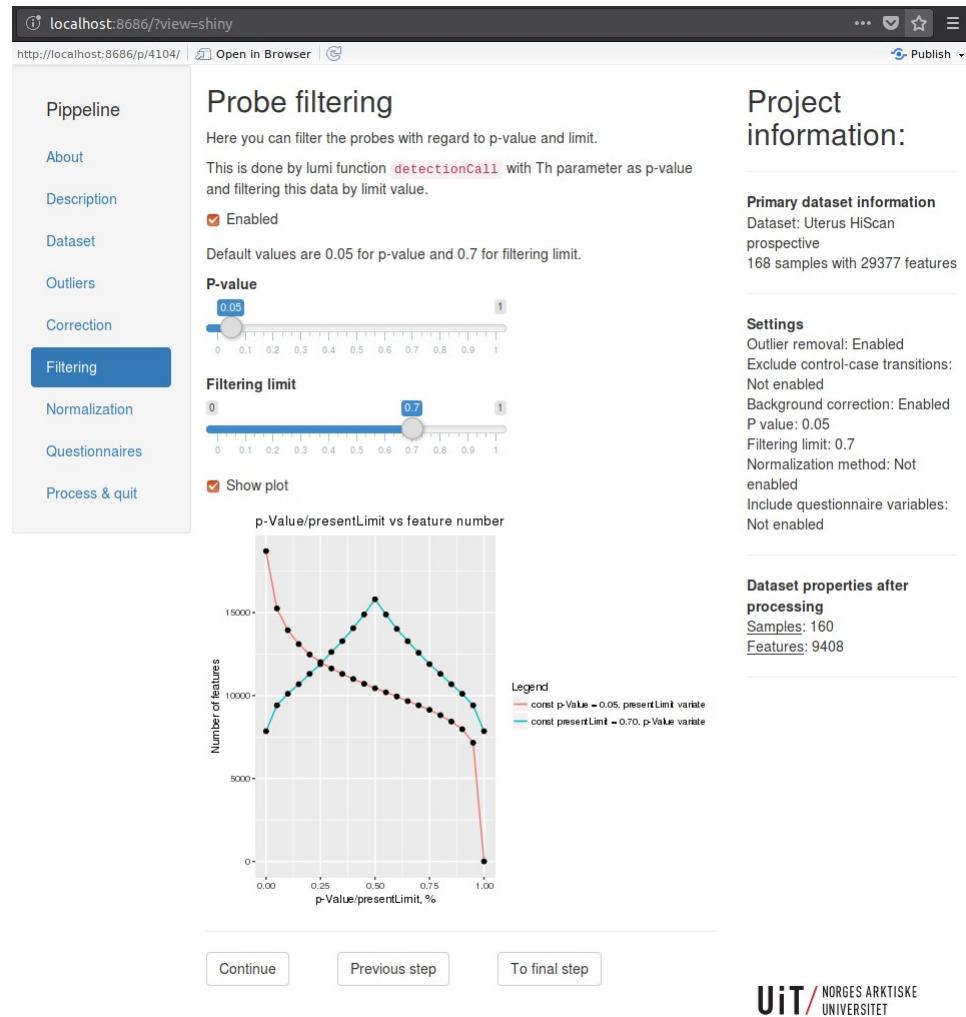


Figure 2.3: A screenshot of the web-interface of Pipeline. In the screenshot users can define at what level they want to filter out probes in the gene expression dataset. Users can define that the output dataset will only include gene expression probes that are present in a percent of the observation.

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[47] and knitr there are few reasons for decoupling analysis code with the presentation of the results through reports or scientific papers. Doing so ensures the correctness 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 pre-processing and data delivery of datasets to researchers. One example of a report is the analyses done in [28] where we documented the association between PAX6 gene expression and PAX6 target genes. Through a simple R script we could share the results and underlying analyses.

Version control everything. Both code and data changes over the course of a research project. Version control everything to make it possible to retrace 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 researchers 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 of archiving development decisions such as choosing a normalization method.

In the NOWAC study we collaborate on data analysis through a self-hosted Gitlab[48] installation. We also open-source code on Github.

2.6 Discussion

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. 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[4] and the Comprehen-

sive R Archive Network (CRAN)[49] provide online hosting for many packages, and users can mix and match these packages to fit their need.

There are many approaches to store and analyze biological datasets. Our approach is targeted towards R users by building an R package, but as we demonstrate in the next chapters, we have designed systems that interface directly with data and analyses from outside the R ecosystem.

One major drawback with the implementation of our approach in the nowac R package is its size. While microarray datasets are relatively small compared to sequencing data, when these datasets grow in number the total size grows as well. This will impact the compile time for the R package, and also its size when it is shared with other researchers. This is something we hope to address in future work.

2.7 Conclusion

In summary, we believe that there are four general rules toward reproducible analyses. We believe that they apply to both our research study and other studies:

- Document and version control datasets and analysis code within the study.
- Share datasets and analysis code through statistical packages.
- Share and report findings through reproducible data analysis reports.
- Standardize and document common data preprocessing and wrangling steps.

In this chapter we have demonstrated one approach that we believe help researchers follow these to enable reproducible research.

/3

Interactive Data Exploration Applications

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. Explorative analysis is essential for understanding biological functions in high-throughput biological datasets. Applications that provide interactive interfaces and visualizations can help researchers study the datasets to discover emerging patterns.

Analysing high-throughput biological datasets require specialized analysis software. Such software is usually written in statistical programming languages, e.g., Python or R, which provide a wealth of statistical packages and frameworks. However, these specialized programming environments often do not provide interactive interfaces for researchers that want to explore the results from the analyses without using a programmatic interface. Frameworks such as Shiny[7] and OpenCPU[8] allow application developers to build systems to interactively explore results from statistical analyses. These are useful for building exploration applications that integrate with statistical analyses written in the R programming language. 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. For visually

exploring biological data there are a range of tools, such as Cytoscape[15] and Circos[16], that support importing an already-analyzed dataset to visualize and browse the data. One problem with these are that they are decoupled from the analysis, making it difficult to retrace the data processing prior to the end results.

The main issue for developing these types of data exploration applications is that they require the integration of disparate systems and tools. The datasets require specialized analysis software, often with large computational resources, and the end users require simple point-and-click interface available on their device. In addition it is crucial for reproducibility to keep track of the underlying data processing steps that were used to generate end visualizations.

We have developed two data exploration applications, Kvik Pathways[19] and MIxT[20, 23] for exploring transcriptional profiles in the NOWAC study through interactive visualizations integrated with biological databases. We first developed Kvik Pathways to explore transcriptional profiles in the context of biological pathway maps. It is a three-tiered web application consisting of three central components, that we later refactored into three separate microservices for use in other applications. These three microservices make up the SMEs in our approach for building data exploration applications. With these microservices we implemented the MIxT web application, and generalized our efforts into general design principles for data exploration applications.

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. We also provide an evaluation that shows that our approach is suitable for this type of interactive applications.

The rest of the chapter is organized as follows: First we present the two motivating use cases for our applications. We then detail the requirements for these types of interactive applications. Following the requirements we detail the Kvik Pathways application, including its architecture and implementation. We then show how we use this first application to generalize its design principle and show we can use them to build applications that follow the SME approach. Following is a description of the implementation of the SMEs approach in the microservices in Kvik. We present how we used these to develop the MIxT web application. Finally we discuss our approach in context of related work, and provide a conclusion.

3.1 Motivating Examples

The need for interactive applications has come from two different studies in the NOWAC project. Both of these rely on advanced statistical analyses and produce comprehensive results that are interpreted by researchers through integration with information from online databases and interactive visualizations. The end results are typically large tables that require manual inspection. Below we describe the two applications before we detail 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 to explore the results from a previous published project ([22], 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 were 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 Tumor-Blood Interactions in Breast Cancer Patients

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.[23] We generated and analyzed expression profiles from blood and matched tumor cells in 173 breast cancer patients included in the 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.

Familiar The applications should use familiar visual representations to present information 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 expression 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 biological processes the genes are a part of.

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 backend layer contains information about pathways and genes, as well as computational and storage resources to process genomic data such as the NOWAC data repository. We have used the packages in Kvik to develop the backend layer. These are discussed in detail in Section 3.5.

The Data Engine in the backend 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).

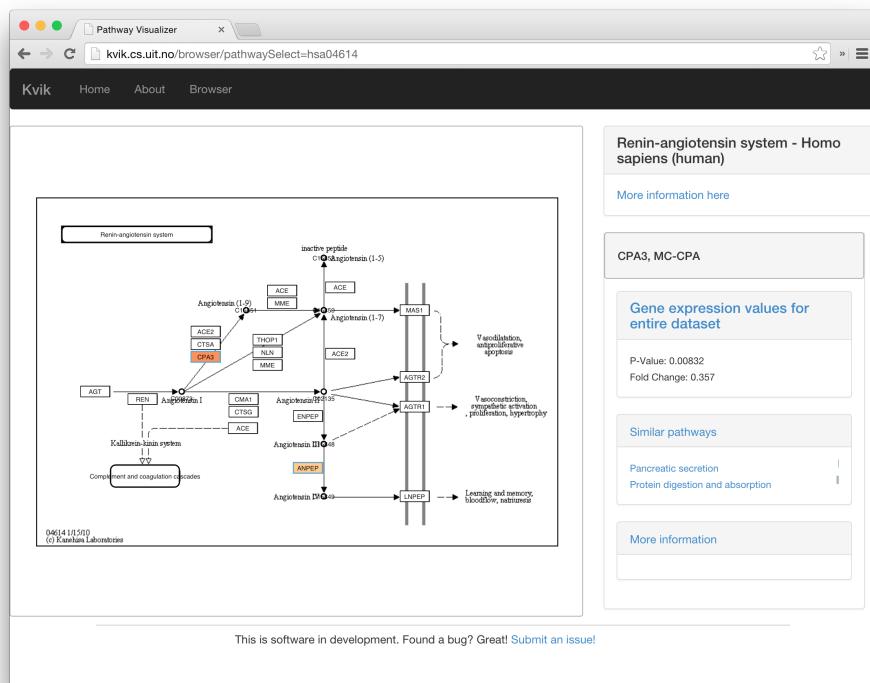


Figure 3.1: Screenshot of the renin-angiotensin pathway (KEGG pathway id hsao4614) in Kvik Pathways. Researchers can visually explore the pathways and read relevant information about genes in the right-hand panel.

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

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

3.3.3 Implementation

To create pathway visualizations the Kvik backend retrieves and parses the KEGG Markup Language (KGML) representation and pathway image from KEGG databases through its REST API.[50] 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 generates a JSON representation that Kvik Pathway uses to create pathway visualizations.

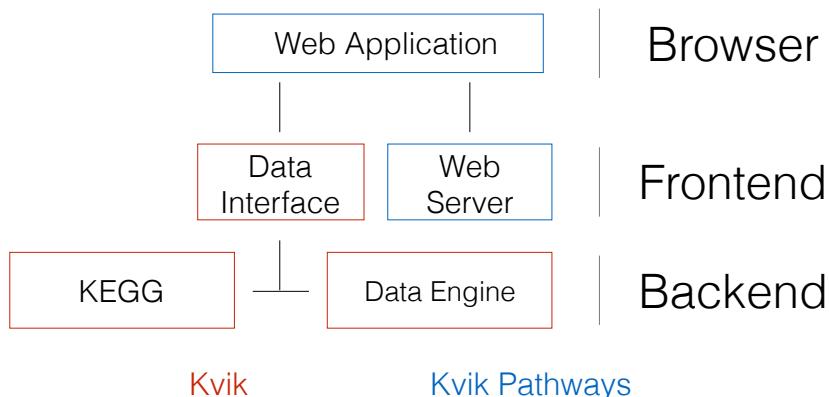


Figure 3.2: The three-tiered architecture of Kvik Pathways.

Kvik Pathways uses Cytoscape.js[51] 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 backend 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 backend 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.

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 Use Case: Analysis of Renin-Angiotensin Pathway

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 [22]). 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 in 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 users to specify and run statistical analysis pipelines in R. Kvik also contains a Javascript package for visualizing KEGG pathways using d3.[52] In addition it provides an interface with online databases such as MsigDB[53] and KEGG[54].

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[8] 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[55], MSigDB, HGNC[56], and KEGG.

Both the compute and the databases service in Kvik build on the standard *net/http* package in the Go programming language.¹ The database service use the *gocache*² package to cache any query to an online database. In addition we deploy each service as Docker containers.³

1. golang.org

2. github.com/fjukstad/gocache.

3. Available at hub.docker.com/r/fjukstad/kvik-r and hub.docker.com/r/fjukstad/db.

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 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 Sigma.[57] 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**).

For the original analyses we built an R package, mixtR,⁴ 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 informa-

4. Available online at github.com/vdumeaux/mixtR.

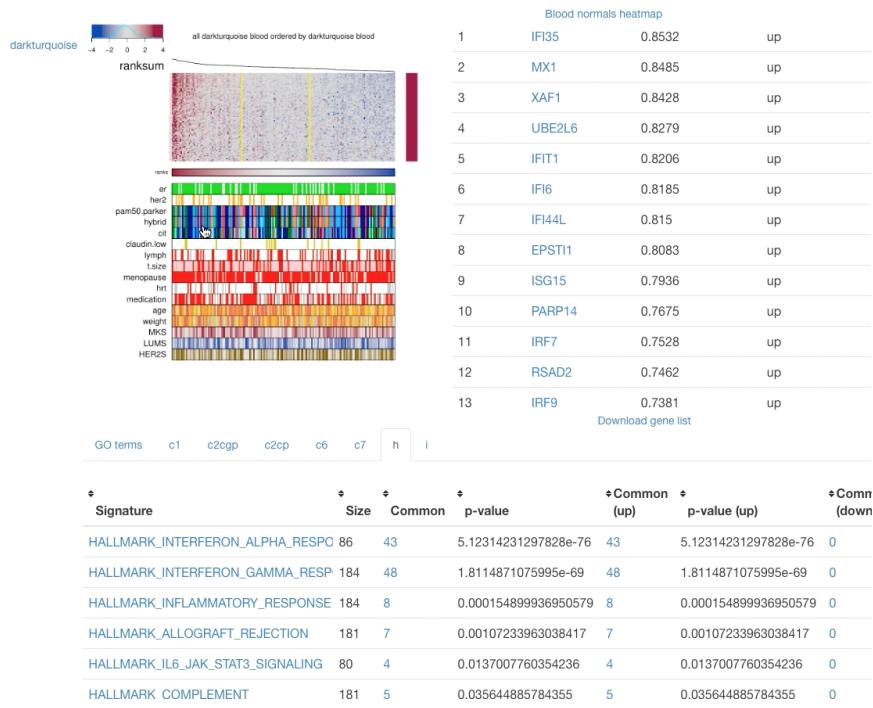


Figure 3.3: MiXT module overview page. The top left panel contains the gene expression heatmap for the module genes. The top right panel contains a table of the genes found in the module. The bottom panel contains the results of gene overlap analyses from the module genes and known gene sets from MSigDB.

tion from biological databases. Each of these components run within Docker containers making the process of deploying the application simple.

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

3.6.4 Evaluation

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

To evaluate the database service we measure the query time for retrieving information about a specific gene with and without caching.⁵ This illustrates

5. More details online at github.com/fjukstad/kvik/tree/master/db/

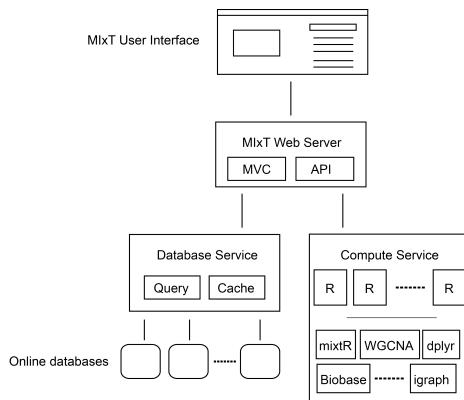


Figure 3.4: The architecture of the MiXT system. It consists of a web application, the hosting web server, a database service for retrieving metadata and a compute service for performing statistical analysis. Note that only the web application and the R package are specific to MiXT, the rest of the components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

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*⁶ 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.

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

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 return the resulting visualization.⁷ 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 OpenCPU⁸ alternative. We

benchmark.

6. See aws.amazon.com/ec2/instance-types for more information about AWS EC2 instance types.

7. More details at github.com/fjukstad/kvik.

8. Built using the *opencpu-server* Docker image.

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

3.6.5 Tumor epithelium-stroma interactions in breast cancer

In addition to the MIxT web application for exploring the link between breast tumor and primary blood, we have also deployed a web application that investigates the link in another dataset[58] We have deployed the application online at `mixt-tumor-stroma.bci.mcgill.ca`. The web application is identical except that the R package accesses another dataset.

3.7 Related Work

OpenCPU is a system for embedded scientific computing and reproducible research.[8] 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.⁹. 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.

9. opencpu.org/apps.html.

Shiny is a web application framework for R¹⁰. 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.[59] 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.[60] Through a Cytoscape App, cyREST, it allows external network creation and analysis through a REST API[61], 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.js¹¹, 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 online repository with a wide range components for visualizing biological data contributed by the bioinformatics community.[18] BioJS builds on node.js¹² 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 air:bit

We have also used the microservice architecture in an application where users can upload and explore air pollution data from Northern Norway.[29] 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.

10. shiny.rstudio.com.

11. js.cytoscapejs.org.

12. nodejs.org.

3.8 Discussion

In this chapter we have given a description of how we successfully built two data exploration applications for high-throughput biological datasets. We have iteratively developed these and ended up with an approach that allows us to develop applications from disparate systems.

There are different arguments for reusing and sharing microservices over libraries in bioinformatics applications, that would justify the cost of hosting and 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 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.

3.9 Future Work

We hope to continue development on applications for interactively exploring biological datasets. Through our approach, and especially the interface to R, we are now able to develop applications that can use any function or retrieve datasets from any R package. This includes the `nowac` package in Chapter 2. We believe that there is a large potential in the available datasets, and that researchers would benefit from being able to interactively explore these.

3.9.1 MIxT

We intend to address few points 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 work-

ing 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[62] 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[63] or SparkR[64] packages to run analyses on top of Spark.[65] Using Spark as an execution engine for data analyses will enable applications to explore even larger datasets.

3.10 Conclusion

We have designed an approach for building data exploration applications in systems biology. We first implemented Kvik Pathways, a web application for exploring a gene expression dataset in the context of pathway maps. We used our experiences to generalize our efforts into a set of central components that these types of applications require. Further we realized these in our SME approach implemented as a set of microservices. Using these services we have built a web application, MIxT, 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.[24] 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

Precision medicine uses patient-specific molecular information to diagnose and categorize disease to tailor treatment to improve health outcome.[37] Important goals 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.[38] Cancer institutions are therefore now offering dedicated personalized medicine programs.

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 time-consuming and thereby a costly part of the experiment. This is because of the manual efforts to set up, analyze, and maintain the analysis pipelines. These pipelines consist of many steps that transform raw data into interpretable results.[21] 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.[66] 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.[67]

Improperly developed analysis pipelines for precision medicine may generate inaccurate results, which may have negative consequences for patient care.[1] When developing analysis pipelines for use in precision medicine it is therefore necessary to track pipeline tool versions, their input parameters, and data. Both to thoroughly document what produced the final clinical reports, and to iteratively improving the quality of the pipeline during development. 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 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.

4.1.1 Initial Data Analysis Pipeline

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 under version control with *git* to track changes as we developed the analysis pipeline. The

pipeline consisted of tools including picard[68], fastqc[69], trimmomatic[70], and the GATK.[71] 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 workbenches such as Galaxy[72] 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 off-site 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.

- Datasets and databases should be under version control 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.
- The specific pipeline tools should also be kept available for later use. Often in bioinformatics, just installing a tool is a time-consuming process because of their many dependencies.
- 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.
- 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.
- 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 analyze 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 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[73] to provide reproducible execution environments, and interface with git together with `git-lfs`[46] 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[74] or build containers

with their own tools. We integrate with the current workflow using git to version control analysis scripts, and use `git-lfs` for versioning of datasets as well. We have designed the pipeline description format to resemble 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.

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[75] 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.[76]

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 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 which is under version control.

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 under version control automatically by `walrus` when new data is produced by the pipeline. We track changes at file granularity.

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 to 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`. Once `walrus` has started to track output datasets, users can use regular `git` commands to inspect its version history. 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. to ensure that genomic variants are called with a desired sensitivity and specificity. To reconfigure an existing pipeline users make the applicable changes to the pipeline description and re-run it with `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 Results

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)[25] to discover SNPs, genomic variants and somatic mutations. We interactively developed a pipeline description that follows the best-practices of The Broad Institute¹ 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

1. Online at software.broadinstitute.org/gatk/best-practices.

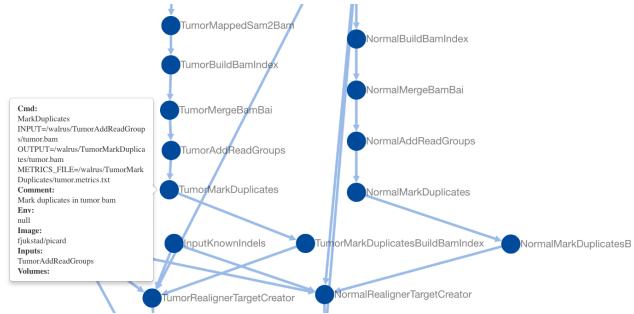


Figure 4.1: Screenshot of the web-based visualization in *walrus*. The user has zoomed in to inspect the pipeline step which marks duplicate reads in the tumor sequence data.

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. 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 [77] 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. 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.

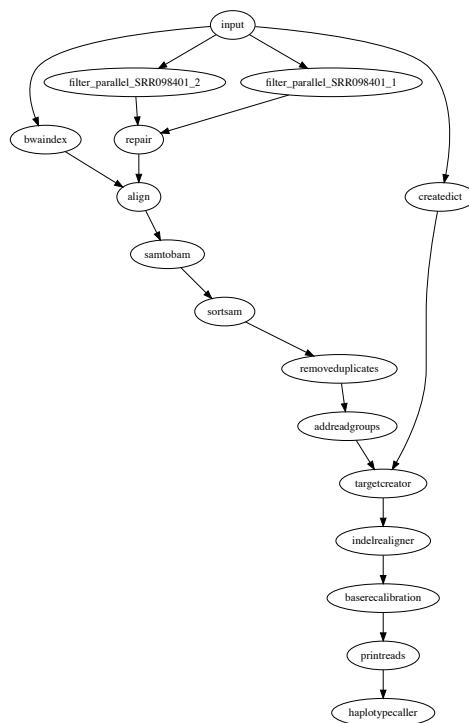


Figure 4.2: In addition to the web-based interactive pipeline visualization, walrus can also generate DOT representations of pipelines. The figure shows the example variant calling pipeline we used in the performance evaluation.

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.

Reproducing results from a scientific publication can be a difficult task. For example, troublesome text formatting on the online version of the reported variant calling pipeline in [77] led to some pipeline tools failing. 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 the example variant-calling pipeline developed 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.^[6] 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 under version control and shared between projects. There are multiple implementations of CWL workflow platforms, e.g. the reference implementation `cwl_runner`^[13], Arvados^[78], Rabix^[79], Toil^[14], Galaxy^[72], and AWE.^[80] 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.^[72] It is possible to run Galaxy on local compute clusters, in the cloud, or using the online Galaxy site.² 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.³ We chose not to use Galaxy because

2. Available at usegalaxy.org.

3. An alpha version of Galaxy with CWL support is available at github.com/common-workflow-language/galaxy.

of missing command-line and scripting support, along with little support for running workflows with different configurations.[3] 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. An interesting project that uses CWL in production is The Cancer Genomics Cloud[81]. They currently support CWL version 1.0 and are planning on integrating Rabix as its CWL executor. 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 implementations. 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.[5] 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.[82] 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 our current practice of using git for versioning.

As discussed in [27], recent projects propose to use containers for life science research. The BioContainers and Bioboxes[83] projects address 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 Virtual Machines (VMS).[84] Both forms of virtualization techniques introduce overhead in I/O-intensive workloads, especially in VMS, but introduce negligible CPU and memory overhead. For precision medicine pipelines the overhead of Docker containers will be negligible since these tend to be compute-intensive and they typically run for several hours.[84] 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.[85]

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 the 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 pre-processing (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 in our results it added 5% in runtime. This makes the entire analysis pipeline slower, but we argue that having the files under version control outweigh the runtime. In addition, there are only a few public `git-lfs` hosting platforms for datasets larger than a few gigabytes, making it necessary to host these in-house. In-house hosting may also be a requirement at different medical institutions.

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.

We believe that future data analysis systems for precision medicine will follow the lines of our proposed approach. Software container solutions provide valuable information in the reporting of the analyses, and they impose little performance overhead. Further, the development of container orchestration systems such as Kubernetes is getting wide adoption nowadays, especially in web-scale internet companies. However, the adoption of such systems in a clinical setting depend on support from more tools, and also the addition of new compute infrastructure.

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

How should we design systems for analyzing and exploring the high-throughput datasets that facilitate sharing, reuse, and reproducibility? This dissertation shows that in many cases the solution is to decompose the applications into small entities that communicate using open protocols. This enables the development of unified systems for reproducible exploration and analysis.

While high-throughput datasets and computing systems will undoubtedly evolve, we believe that the SME approach proposed here can offer a new perspective on developing applications for exploring and analyzing biological data. We hope that our approach can steer the development of bioinformatics applications away from large monolithic applications to applications composed of diverse systems. We believe that this approach can help the community develop new systems to faster meet the needs of the upcoming biological dataset analyses.

In Chapter 2 we show an approach to store the biological data and analysis code from a complex epidemiological study in a shareable software package. We show how we explicitly track versions of code and data, and how we can generate reproducible data analysis reports for the processed datasets. We believe that future studies can benefit from applying our approach, and that future advances in research is dependent on sharing of both datasets and analysis code. In chapter 3 we show how we can build interactive data exploration applications that interface with these software packages through a microservice architecture. We have implemented this approach through the

microservices in *Kvik*. We show that this architecture style is suitable for building such applications, and have used it to develop the *Kvik Pathways* and *MixT* web applications. These have been successfully used to explore transcriptional profiles in the NOWAC study, especially to investigate the interactions between genes and pathways in the patient tumor and blood cells. We believe that the research community in general will benefit greatly if more projects start to develop their applications using our approach. It promotes the sharing of resources, both computational and storage, and we believe that it is the future for collaborative research. In chapter 4 use the same approach, to compose systems of disparate tools, for developing biological data analysis pipelines, implemented in *walrus*. To ensure reproducible results, we supplement the processing with data versioning to track provenance of the data through the pipeline and across pipeline versions.

Combined, these systems demonstrate the applicability of our approach across a range of different use cases.

In the rest of this chapter we summarize end-to-end lessons learned during this work. We then discuss the work in the context of research and in the clinic, before we propose areas for future work.

5.1 Lessons Learned

Through the design of the **SME** approach for analyzing and exploring biological datasets, as well as the different implementations of the approach, we have solved challenges and learned some key lessons.

There is no single solution programming language or system. In the field of bioinformatics there have been tremendous efforts to develop analysis tools for improving the analysis of new biological datasets. This has led to systems being written in a plethora of different languages, and deployed on top of different systems. This the main motivation behind our **SME** approach together with software containers.

Take advantage of existing tools. The ability to develop applications for analyzing biological datasets comes from the availability of existing tools. By developing easy-to-use interfaces for the existing tools, it is possible to develop new applications without reimplementing key features.

Simplicity is key. When proposing a new approach for either managing datasets, writing data exploration applications, or developing analysis pipelines, it is not possible to overstate the importance of the simplicity of the solution.

Through our experiences we have tried different

Researchers are not software engineers. When designing a new approach to store and analyze high-throughput biological datasets, it is important to remember that its users have limited software engineering backgrounds. Especially when the implementation is based on complex systems such as git, the learning curve for the system is steep and require training of its users. In our project we have organized workshops in both R and git to get the researchers in the NOWAC study comfortable with these systems.

5.2 Future Work

As we have discussed in previous chapters, there are some limitations to our approach and its implementations. To summarize these, the main areas for improvement are:

- **Versioning of datasets:** git was not designed to version large binary files, such as biological datasets, and it does not provide the required performance or scalability to version the large biological data.
- **Additional evaluation:** while we have shown that the SME approach can be used to develop systems for managing research data, developing interactive applications and data analysis pipelines, we would like to additionally understand its performance and scalability.
- **Refactoring and test coverage:** while we provide fully implemented solutions for data storage, interactive applications, and data analysis pipelines, they all have areas of improvement with regards to performance, scalability, and robustness.
- **Distributed execution:** while walrus orchestrate execution of Docker containers, we do not support the execution of these on multiple compute nodes. Distributing the computation on multiple machines will reduce the execution time if we can share the data across the machines efficiently. We would also like to evaluate the possibility of using an existing container orchestration system, such as Kubernetes, to orchestrate the execution of an analysis pipeline. Many of these already provide functionality for distributed execution of software containers.
- **Wide adoption of a pipeline description format:** we are not the first to

propose a new computing standard.¹ We found that the current standards were either too verbose, or did not require the use of software containers, which we believe are instrumental to make the pipeline reproducible. However, we recognize the need for a single open standard, and hope to contribute to its development.

We aim to refine and continue development on our SME approach to address these challenges, and that we can inspire a more unified development community in bioinformatics. We believe that the future of cancer research relies on the successful integration of the diverse data analysis and storage systems from the many research institutions, to take advantage of the abundance of available resources. It will require open interfaces between the systems, and that the systems can correctly document all local data processing steps to ensure reproducible results. This will definitely continue to be an interesting area of research.

¹. xkcd.com/927

Bibliography

- [1] S. Roy, C. Coldren, A. Karunamurthy, N. S. Kip, E. W. Klee, S. E. Lincoln, A. Leon, M. Pullambhatla, R. L. Temple-Smolkin, K. V. Voelkerding *et al.*, “Standards and guidelines for validating next-generation sequencing bioinformatics pipelines: A joint recommendation of the association for molecular pathology and the college of american pathologists,” *The Journal of Molecular Diagnostics*, 2017.
- [2] 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.
- [3] 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.
- [4] R. C. Gentleman, V. J. Carey, D. M. Bates, B. Bolstad, M. Dettling, S. Dudoit, B. Ellis, L. Gautier, Y. Ge, J. Gentry *et al.*, “Bioconductor: open software development for computational biology and bioinformatics,” *Genome biology*, vol. 5, no. 10, p. R80, 2004.
- [5] Pachyderm, <http://pachyderm.io>.
- [6] 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.
- [7] Shiny, <http://shiny.rstudio.com>.
- [8] J. Ooms, “The opencpu system: Towards a universal interface for scientific computing through separation of concerns,” *arXiv preprint arXiv:1406.4806*, 2014.

- [9] E. S. Raymond, *The art of Unix programming*. Addison-Wesley Professional, 2003.
- [10] T. Erl, *Service-oriented architecture*. Prentice hall New York, 2005, vol. 8.
- [11] I. Nadareishvili, R. Mitra, M. McLarty, and M. Amundsen, *Microservice Architecture: Aligning Principles, Practices, and Culture*. " O'Reilly Media, Inc.", 2016.
- [12] J. Köster and S. Rahmann, "Snakemake—a scalable bioinformatics workflow engine," *Bioinformatics*, vol. 28, no. 19, pp. 2520–2522, 2012.
- [13] P. Amstutz, M. R. Crusoe, N. Tijanić, B. Chapman, J. Chilton, M. Heuer, A. Kartashov, D. Leehr, H. Ménager, M. Nedeljkovich *et al.*, "Common workflow language, v1. 0," 2016.
- [14] 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.
- [15] 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.
- [16] M. Krzywinski, J. Schein, I. Birol, J. Connors, R. Gascoyne, D. Horsman, S. J. Jones, and M. A. Marra, "Circos: an information aesthetic for comparative genomics," *Genome research*, vol. 19, no. 9, pp. 1639–1645, 2009.
- [17] E. Lund, V. Dumeaux, T. Braaten, A. Hjartåker, D. Engeset, G. Skeie, and M. Kumle, "Cohort profile: the norwegian women and cancer study—nowac—kvinner og kreft," *International journal of epidemiology*, vol. 37, no. 1, pp. 36–41, 2007.
- [18] 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.
- [19] 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.

- [20] 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.
- [21] Y. Diao, A. Roy, and T. Bloom, “Building highly-optimized, low-latency pipelines for genomic data analysis.” in *CIDR*, 2015.
- [22] 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.
- [23] 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.
- [24] B. Fjukstad, V. Dumeaux, M. Hallett, and L. A. Bongo, “Reproducible data analysis pipelines for precision medicine,” <https://www.biorxiv.org/content/early/2018/06/25/354811>, 2018.
- [25] 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.
- [26] B. Fjukstad and L. A. Bongo, “A review of scalable bioinformatics pipelines,” *Data Science and Engineering*, vol. 2, no. 3, pp. 245–251, 2017.
- [27] I. A. Raknes, B. Fjukstad, and L. Bongo, “nsroot: Minimalist process isolation tool implemented with linux namespaces,” *Norsk Informatikkonferanse*, 2017.
- [28] 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.
- [29] 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.

- [30] J. D. Watson, F. H. Crick *et al.*, “Molecular structure of nucleic acids,” *Nature*, vol. 171, no. 4356, pp. 737–738, 1953.
- [31] 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.
- [32] I. H. G. S. Consortium *et al.*, “Initial sequencing and analysis of the human genome,” *Nature*, vol. 409, no. 6822, p. 860, 2001.
- [33] M. L. Metzker, “Sequencing technologies—the next generation,” *Nature reviews genetics*, vol. 11, no. 1, p. 31, 2010.
- [34] S. D. Kahn, “On the future of genomic data,” *science*, vol. 331, no. 6018, pp. 728–729, 2011.
- [35] M. Baker, “Why scientists must share their research code,” *Nature News*, 2016.
- [36] “Reproducibility in cancer biology: The challenges of replication,” *eLife*, vol. 6, p. e23693, jan 2017.
- [37] 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.
- [38] I. F. Tannock and J. A. Hickman, “Limits to personalized cancer medicine,” *N Engl J Med*, vol. 375, no. 13, pp. 1289–1294, 2016.
- [39] V. Dumeaux, K. S. Olsen, G. Nuel, R. H. Paulssen, A.-L. Børresen-Dale, and E. Lund, “Deciphering normal blood gene expression variation—the nowac postgenome study,” *PLoS genetics*, vol. 6, no. 3, p. e1000873, 2010.
- [40] M. Holden, L. Holden, K. Olsen, and E. Lund, “Local in time statistics for detecting weak gene expression signals in blood — illustrated for prediction of metastases in breast cancer in the nowac post-genome cohort,” *Advances in Genomics and Genetics*, vol. 55, no. unknown, pp. 11–28, 2017.
- [41] V. Dumeaux and E. Lund, “Gene expression profile in diagnostics,” Oct. 22 2015, uS Patent App. 14/646,010.

- [42] Y. Xie, *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, 2016.
- [43] git-submodule, <https://git-scm.com/docs/git-submodule>.
- [44] git-raw, <https://github.com/atofigh/git-raw>.
- [45] git-annex, <https://git-annex.branchable.com>.
- [46] Git LFS, <https://git-lfs.github.com>.
- [47] R Markdown, <http://rmarkdown.rstudio.com>.
- [48] Gitlab, <https://gitlab.com/>.
- [49] The Comprehensive R Archive Network (CRAN), <https://cran.r-project.org>.
- [50] M. Tanabe and M. Kanehisa, “Using the kegg database resource,” *Current protocols in bioinformatics*, vol. 38, no. 1, pp. 1–12, 2012.
- [51] M. Franz, C. T. Lopes, G. Huck, Y. Dong, O. Sumer, and G. D. Bader, “Cytoscape.js: a graph theory library for visualisation and analysis,” *Bioinformatics*, vol. 32, no. 2, pp. 309–311, 2015.
- [52] M. Bostock, V. Ogievetsky, and J. Heer, “D³ data-driven documents,” *IEEE transactions on visualization and computer graphics*, vol. 17, no. 12, pp. 2301–2309, 2011.
- [53] A. Liberzon, A. Subramanian, R. Pinchback, H. Thorvaldsdóttir, P. Tamayo, and J. P. Mesirov, “Molecular signatures database (msigdb) 3.0,” *Bioinformatics*, vol. 27, no. 12, pp. 1739–1740, 2011.
- [54] M. Kanehisa and S. Goto, “Kegg: kyoto encyclopedia of genes and genomes,” *Nucleic acids research*, vol. 28, no. 1, pp. 27–30, 2000.
- [55] E. Sayers, “Entrez programming utilities help,” <http://www.ncbi.nlm.nih.gov/books/NBK25499>, 2009.
- [56] K. A. Gray, B. Yates, R. L. Seal, M. W. Wright, and E. A. Bruford, “Genenames. org: the hgnc resources in 2015,” *Nucleic acids research*, vol. 43, no. D1, pp. D1079–D1085, 2014.
- [57] Sigma, <http://sigmajs.org>.

- [58] B. J. Boersma, M. Reimers, M. Yi, J. A. Ludwig, B. T. Luke, R. M. Stephens, H. G. Yfantis, D. H. Lee, J. N. Weinstein, and S. Ambs, “A stromal gene signature associated with inflammatory breast cancer,” *International journal of cancer*, vol. 122, no. 6, pp. 1324–1332, 2008.
- [59] 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.
- [60] 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.
- [61] 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.
- [62] AppArmor, <http://wiki.ubuntu.com/AppArmor>.
- [63] sparklyr: R interface for Apache Spark, <http://spark.rstudio.com>.
- [64] SparkR, <http://spark.apache.org/docs/latest/sparkr.html>.
- [65] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets.” *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.
- [66] 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.
- [67] A. Sboner and O. Elemento, “A primer on precision medicine informatics,” *Briefings in bioinformatics*, vol. 17, no. 1, pp. 145–153, 2015.
- [68] Picard, <https://broadinstitute.github.io/picard>.
- [69] S. Andrews *et al.*, “Fastqc: a quality control tool for high throughput sequence data,” 2010.
- [70] A. M. Bolger, M. Lohse, and B. Usadel, “Trimmomatic: a flexible trimmer for illumina sequence data,” *Bioinformatics*, vol. 30, no. 15, pp. 2114–2120, 2014.

- [71] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kerntsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly *et al.*, “The genome analysis toolkit: a mapreduce framework for analyzing next-generation dna sequencing data,” *Genome research*, vol. 20, no. 9, pp. 1297–1303, 2010.
- [72] 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.
- [73] Docker, <https://www.docker.com>.
- [74] BioContainers, “Biocontainers,” <https://biocontainers.pro>, 2017, [Online; Accesssed: 16.08.2017].
- [75] 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.
- [76] J. Ellson, E. Gansner, L. Koutsofios, S. C. North, and G. Woodhull, “Graphviz—open source graph drawing tools,” in *International Symposium on Graph Drawing*. Springer, 2001, pp. 483–484.
- [77] 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.
- [78] Arvados, “Arvados | open source big data processing and bioinformatics,” <https://arvados.org>, 2017, [Online; Accesssed: 16.08.2017].
- [79] 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.
- [80] 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.
- [81] J. W. Lau, E. Lehnert, A. Sethi, R. Malhotra, G. Kaushik, Z. Onder, N. Groves-Kirkby, A. Mihajlovic, J. DiGiovanna, M. Srdic *et al.*, “The cancer genomics

cloud: Collaborative, reproducible, and democratized—a new paradigm in large-scale computational research,” *Cancer research*, vol. 77, no. 21, pp. e3–e6, 2017.

- [82] Kubernetes, <https://kubernetes.io>.
- [83] 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.
- [84] 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.
- [85] C. Boettiger, “An introduction to docker for reproducible research,” *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, pp. 71–79, 2015.