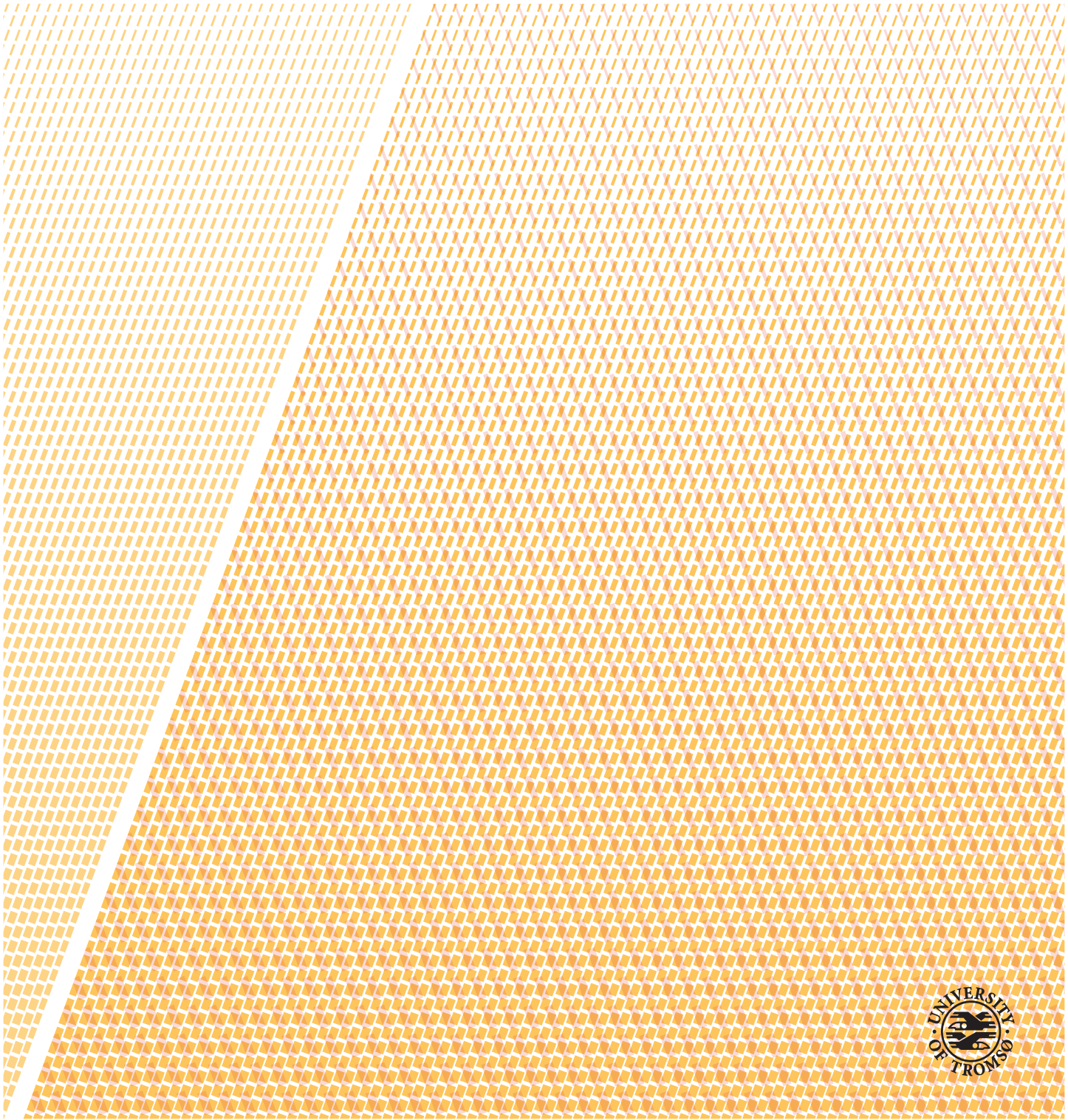
UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

Faculty of Science and Technology
Department of Computer Science

A Unified Approach for Reproducible Analysis of High-Throughput Biological Datasets

Bjørn Fjukstad

A dissertation for the degree of Philosophiae Doctor



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

–Ivar Tollefsen

Abstract

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

This dissertation argues that, instead, we can design a unified approach that integrates disparate systems and data into fully reproducible biological data analysis frameworks. In particular, we show how software container technologies together with well-defined interfaces, configurations, and orchestration provide the necessary foundation to build reproducible analysis pipelines for biological datasets, as well as highly interactive data exploration applications.

We show the need and feasibility of our approach through a number of different applications for analyzing and exploring biological datasets. These applications were developed to meet the requirements of researchers in systems epidemiology and precision medicine. We evaluate the approach through these systems using real datasets. Our results show that our approach can be used to enable reproducible data analysis and exploration of high-throughput biological datasets while still providing the performance of related systems.

Acknowledgements

Insert acks here.

Contents

List of Figures

List of Tables

1

Introduction

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

There are several computational challenges for researchers to analyze and explore biological datasets. These challenges are common for large datasets such as high-throughput sequencing data that require long-running, deep analysis pipelines, as well as smaller datasets, such as microarray data, that require complex, but short-running analysis pipelines. The first is the time and knowledge required to find and set up the necessary analysis tools to start analyzing a modern biological dataset. The second is ensuring the correct input parameters, tool versions, database versions, and dataset versions when

analyzing, and reporting analysis results to enable reproducible science. A third challenge is efficiently exploring the results of the analyses interactively. This includes developing tools that can efficiently visualize the heterogeneous datsets and integrate them with known biology from databases to provide necessary information for interpreting the results. The final challenge is reusing the analysis pipelines and exploration tools with new datasets, methods, and research questions.

As a result, there are a wealth of specialized approaches and systems to enable analysis of the complex biological data. To develop deep analysis pipelines in bioinformatics, Galaxy[1] has for a long time provided a simple interface to set up and execute analysis pipelines for genomic datasets. However, the Galaxy system is less effective for explorative and flexible analyses where it is necessary to try out different tools with different configurations.[2] New initiatives such as the Common Workflow Language (CWL) provide users a standardized way of describing and sharing an analysis pipeline, and has multiple implementations such as the reference implementation cwl_runner,[1] Arvados,[3] Rabix,[4] Toil,[5] Galaxy,[1] and AWE.[6] While these systems provide a viable option for batch-processing of large biological datasets, researchers with smaller datasets at hand, can analyze and explore them through interactive languages and interpreters such as Python or the R programming language. Through the package repository Bioconductor, there are a wide range of R packages to analyze biological datasets. These include tools for both analyzing and visualizing the datasets. For users with little or no programming experience it is possible to access and explore datasets thorough applications using the Shiny or OpenCPU frameworks. These let developers write applications in the R programming language, and users can access and explore the data through web applications. Standalone systems such as Cytoscape provide a specialized software platform to visualize and explore complex biological datasets.[**?**] Generalized systems for analyze a wide range of big datasets are now starting to get attention in bioinformatics. An example of one of these is Pachyderm, a system for deploying and managing multi-stage, lagnuage-agnostic data pipelines.[**?**] In addition to tracking pipeline configrations, it also provides full provenance for the data. Another example is Aparche Spark, an analytics engine for large-scale data processing.[**?**]. Both of these provide useful abstractions for reproducible analyses of large-scale datasets, but they have yet to see wide-spread adpotion in Bioinformatics. 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 design a unified approach that integrates disperate systems and data into fully reproducible biological data anal-

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

ysis frameworks. In particular, we show how software container technologies together with well-defined interfaces, configurations, and orchestration provide the necessary foundation to build reproducible analysis pipelines for biological datasets, as well as highly interactive data exploration applications.

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

- It enables reproducible research by packaging applications and tools within containerized environments. This enables sharing of tools and simplifies the tedious task of installing specific tools.

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

- It enables applications to use tools written in any programming language, using open standards to communicate between tools and systems. This allows for exploration tools to interface with both statistical analyses and biological databases.

- It facilitates the development of flexible and configurable systems by separating appliications and tools into small composable parts. This allows developers to reuse parts of a system to fit new methods and datasets.

From collaboration with researchers in systems epidemiology and precision medicine we were asked to develop a set of applications and systems that could enable them to analyze and explore their datasets. From these systems we extraplolated a set of general design principles to form a unified approach. We implement our approach thorugh a series of applications and tools built on top of a stack of open source systems with software containers as the common foundation. We evaluate the approach through these systems using real datasets and show its viability.

From a longer-term perspective we discuss the general patterns for implementing modern data analysis systems for use in precision medicine and dicuss why our approach is a suitable option. As more datasets are produced every year, research will depend on systems being easy to pick up, and 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 there is a move towards using more sophisticated approaches to analyze biological datasets through workflow and pipeline mangers such as Galaxy[1] and the CWL[**?**]. These simplify setting up the analysis pipeline pipeline, maintaining, and updating it. However, these tools still have their limitations and shell scripts are still the de facto standard building analysis pipelines in bioinformatics. For exploring biological data there are a range of tools, such as Cytoscape[**?**] and Circos[**?**], that support importing an already-analyzed dataset to visualize and browse the data.

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

1. **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.

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

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

4. **Reproducibility:** While there are tools for analyzing most data types today, there is little or no effort to fully document the entire pipeline from raw data to interpretable results. This includes tool versions, parameters, data, and databases. This makes analysis results difficult to reproduce.

Because of these drawbacks, an approach for reproducible data analysis and exploration would have significant benefits for the complex interpretation of biological datasets.

1.2 The X Model/approach/etc.

From the collaboration with researchers we have developed applications for two specific research areas. The first to explore the datasets of a large population-

based research cohort. The second to analyze sequencing datasets for use in a precision medicne setting. Although these areas require widly different systems with different requirements, the systems share common design patterns. We discuss the different areas separately before highlighting the similarities.

Deep analysis pipelines.

Analysis of high-throughput sequencing datasets requires deep analysis pipelines with a large number of steps that transform raw data into interpretable results[7]. There are a large number of tools available to perform the different processing steps, written in a wide range of programming languages. The tools, and their dependencies, can be difficult to install, and they require users to correctly manage a range of input parameters that affects the output results. With these observations in mind we used software containers to package the tools we needed for our analyses, one tool per container image. This made it possible to share the container image between compute systems without installing any dependencies or additional packages. To keep track of input parameters as well as the flow of data in a pipeline we designed a text-based specification for analysis pipelines. This specification includes information such as input parameters and tool versions.

This approach was then implemented 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.

Interactive exploration. Analysis pipelines and workflows typically require researchers to browse and explore the final output. In addition it may be useful to further explore results by modifying analysis parameters to execute new analyses. As with analysis pipelines there are complete exploration tools as well as software libraries to develop custom applications. 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. With a lot modern visualization libraries created for the web there are also possibilities to develop applications that target users on any platform. From these observations we wrote an interface to the R programming language, that would allow us to interface with the wealth of existing software packages 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. In addition, to provide reproducible execution environments we also packaged

into software containers that could be easily deployed and shared.

This approach was then implemented as a part of *Kvik*, a collection of packages to develop new data exploration applications. Kvik allows applications written in any modern programming language to interface with the wealth of bioinformatics packages in the R programming language, as well as information available through online databases. We have used Kvik to develop the Matched Interactions Across Tissues (MIXT) system for exploring and comparing transcriptional profiles from blood and tumor samples in breast cancer patients, in addition to applications for exploring biological pathways.

Parallels. Both approaches to build analysis pipelines and to write interactive data exploration applications build on the same principles. In both areas we break down the systems in to smaller composable units, e.g. a tool, and package these into software containers which are then orchestrated together. These containers are configured and communicate using open protocols that make it possible to interface with them using any programming language. We can keep track of the configuration of the containers and their orchestration using software versioning systems, and provide the necessary information to reproduce analyses or a complete system.

1.3 Systems Implemented with the X Model/approach/etc.

In this section we detail the different systems we have built on top of walrus and Kvik.

The first system we built on top of walrus was a pipeline to analyze a patient's primary tumor and adjacent normal tissue, including subsequent metastatic lesions.[**?**] We packaged the necessary tools for the analyses into software containers and wrote a pipeline description with all the necessary data processing steps. Some of the steps required us to develop specialized scripts to generate customized plots, but these were also wrapped in a container. From the analyses we discovered, among other findings, inherited germline mutations that are recognized to be among the top 50 mutations associated with an increased risk of familial breast cancer.

The second analysis pipeline we implemented was to enable comparison of a RNA-seq dataset to gene expression values collected from the same samples. The pipeline preprocesses the RNA dataset for all samples, and generates transcript quantifications. As with the first pipeline we used existing tools

together with specialized analysis scripts packaged into a container to ensure that we could reproduce the execution environments.

The first interactive data exploration application we built was Kvik Pathways. It allows users to explore gene expression data from the Norwegian Women and Cancer (NOWAC) cohort in the context of interactive pathway maps.[**?**] 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.[8]

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

The second application we built the MIXT web application. A system to explore and compare transcriptional profiles from blood and tumor samples in breast cancer patients. The application is built to simplify the exploration of results from the Matched Interactions Across Tissues (MIxT) study. Its goal was to identify genes and pathways in the primary breast tumor that are tightly linked to genes and pathways in the patient blood cells.[9] The web application interfaces with the methods implemented as an R package and integrates the results together with information from biological databases through a simple user interface.

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

Combined these systems and applications demonstrate how the X approach is useful for both batch processing of datasets, as well as interactive applications.

1.4 Summary of Results

We show the viability of our approach through real-world applications in systems epidemiology and precision medicine. We show that our approach for

building interactive data exploration application, implemented in Kvik, provides the necessary interfaces as well as better performance than other related tools. We show that walrus, the implementation of our approach, is suitable for analyzing precision medicine datasets and we show that the additional steps to track data provenance does not impose .. overhead.

We have used walrus to analyze a whole-exome dataset to from a sample in the McGill Genome Quebec [MGGQ] dataset (GSE58644)[10] 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 can be used to enable reproducible data analysis and exploration of high-throughput biological datasets while still providing the required performance.

1.5 List of papers

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

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	Designed, implemented, and deployed Kvik and Kvik Pathways. Evaluated the system and wrote the manuscript.
Publication date	15 March 2015
Publication venue	F1000
Citation	[11] 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

Title	Building Applications For Interactive Data Exploration In Systems Biology.
Authors	**Bjørn Fjukstad**, Vanessa Dumeaux, Karina Standahl Olsen, Michael Hallett, Eiliv Lund, and Lars Ailo Bongo.
Description	Describes how we further developed the ideas from Paper 1 into an approach that we used to build the MIXT web application.
Contribution	Designed, implemented, and deployed Kvik and the MIXT web application. Evaluated the system and wrote the manuscript.
Publication date	20 August 2017.
Publication venue	The 8th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics (ACM BCB) August 20–23, 2017.
Citation	[12] 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

Title	Interactions Between the Tumor and the Blood Systemic Response of Breast Cancer Patients
Authors	Vanessa Dumeaux, **Bjørn Fjukstad**, Hans E Fjosne, Jan-Ole Frantzen, Marit Muri Holmen, Enno Rodegerdts, Ellen Schlichting, Anne-Lise Børresen-Dale, Lars Ailo Bongo, Eiliv Lund, Michael Hallett.
Description	Describes the MIXT system which enables identification of genes and pathways in the primary tumor that are tightly linked to genes and pathways in the patient Systemic Response (SR).
Contribution	Designed, implemented, and deployed the MIXT web application. Contributed to write the manuscript.
Publication date	28 September 2017.
Publication venue	PLoS Computational Biology
Citation	[9] 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

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	Wrote the manuscript.
Publication date	23 October 2017
Publication venue	Data Science and Engineering 2017.
Citation	[13] B. Fjukstad and L. A. Bongo, "A review of scalable bioinformatics pipelines," *Data Science and Engineering*, vol. 2, no. 3, pp. 245–251, 2017

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

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

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

1.6 Dissertation Plan

This thesis is organized as follows. Chapter 2 describes the characteristics of state-of-the-art biological datasets, the analysis required to extract knowledge from these, and the available tools and analysis frameworks. Chapter 3 describes in detail how we use a container centric development model to build a tool, walrus, to develop and execute deep analysis pipelines. In Chapter 4 we describe how we used the same model to develop applications to interactively explore results from statistical analyses. Finally, Chapter 5 concludes the work and discusses future directions.

2

Modern Biological Data Analysis

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

In this chapter we give a background in the different aspects of analyzing and exploring biological datasets. We highlight the necessary processing steps from data generation and to interpretation of results.

2.1 High-Throughput Datasets in Research and Medicine

DNA sequencing is the process of determining the order of nucleotides within a strand of DNA.

Precision medicine uses patient-specific molecular information to diagnose and categorize disease to tailor treatment to improve health outcome.[19] Important research goal in precision medicine are to learn about the variability of the molecular characteristics of individual tumors, their relationship to outcome, and to improve diagnosis and therapy.[20] International cancer institutions are therefore offering dedicated personalized medicine programs, but while the data collection and analysis technology is emerging, there are still unsolved problems to enable reproducible analyses in clinical settings. For cancer, high throughput sequencing is the main technology to facilitate personalized diagnosis and treatment since it enables collecting high quality genomic data from patients at a low cost.

2.1.1 Norwegian Women and Cancer

The NOWAC systems epidemiology research project is a study designed to identify the possible relationships between lifestyle and the risk of cancer. It started its data collection in 1991. In 2006 the study contained questionnaire information from over 170 000 women. Since then the data collection started in 1998, the NOWAC postgenome biobank has grown to over 60 000 blood samples and 800 biopsies that have been, or will be, analyzed using whole-genome gene expression analysis tools. Additionally the biobank contains information about exposure through questionnaires answered by the participants of the study.

2.2 Preprocessing

2.3 Analysis Pipelines

2.4 Interactive Applications

3

Deep Analysis Pipelines

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

3.1 Use Case and Motivation

For cancer, high throughput sequencing is the main technology to facilitate personalized diagnosis and treatment since it enables collecting high quality genomic data from patients at a low cost. Analyzing sequencing datasets require deep analysis pipelines with a large number of steps that transform raw data into interpretable results.[7] 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.[21] 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.[22]

When developing analysis pipelines for use in precision medicine it is necessary

to track pipeline tool versions, their input parameters, and data. Both to thoroughly document what produced the end results, but also to compare results from different pipeline runs. Because of the iterative process of developing the analysis pipeline, thoroughly investigating emerging patterns and signatures, it is necessary to use analysis tools that facilitates modifying pipeline steps and adding new ones with little developer effort.

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

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

We chose a command-line script over more complex pipelining tools or workbenches such as Galaxy[1] because of its fast setup time on our available compute infrastructure, and familiar runtime. More complex systems could be beneficial in larger research groups with more resources to compute infrastructure maintenance, whereas command-line scripting languages require little to none 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 an analyzed.

After we completed the first round of analyses we summarized our efforts and noted some lessons learned. First, datasets and databases should be version controlled and stored along with the pipeline description. In the analysis script we referenced to datasets and databases by their physical location on a storage system, but these were later moved without updating the pipeline description causing extra work. A solution would be to add the data to the

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

same version control repository hosting the pipeline description. Second, the specific pipeline tools should also be kept available for later use. Since installing many bioinformatics tools require a long list of dependencies, it is beneficial to store the pipeline tools to reduce the time to start analyzing new data or re-run analyses. Another lesson is that it should be easy to add new tools to an existing pipeline and execution environment. This includes installing the specific tool and adding to an existing pipeline. Bundling tools within software containers, such as Docker, and hosting them on an online registry simplifies the tool installation process since the only requirement is the container runtime. Another lesson learned is that while bash scripts have their limitations, using a well-known format that closely resembles the normal command-line use clearly have its advantages. It is easy to understand what tools were used, their input parameters, and the data flow. However, from our experience when these analysis scripts grow too large they become too complex to modify and maintain.

The above problem areas are not just applicable to our research group, but common to other 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 pipelines 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.

3.2 `walrus`

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

pipeline managers require the use of cluster computers or cloud environment, we focus on single compute nodes often found in clinical environments such as hospitals.

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

With walrus we target pipeline developers that use command-line tools and scripting languages to build and run analysis pipelines. Users wrap their existing tools in Docker containers and can use their existing compute infrastructure. We integrate with the current workflow using git to version control analysis scripts, and use git-lfs to expand to versioning of datasets as well. We have designed the pipeline description format resembles the command line syntax as much as possible.

3.2.1 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 3.1 shows an example pipeline stage that uses MuTect[23] to detect somatic point mutations. Users can also specify the tool versions

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, walrus provides a plotting tool for visualizing the pipeline using tools such as Graphviz.[7] before running the pipeline.

Listing 3.1: 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": [
```

5. docker.com
6. git-lfs.github.com
7. graphviz.org

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

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

3.2.2 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 with bioinformatics tools installed 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,[24] 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 the underlying 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 version ID for the pipeline run, which later can be used to investigate a previous pipeline run.

3.2.3 Data Management

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

In walrus we interface with git to track any output file from the analysis pipeline. When users start running a pipeline walrus will look for a git repository in the same directory as the pipeline description, if it cannot find a repository it will create one for the user. As pipeline stages complete, walrus will automatically add and commit the output files for each stage to the git repository, using git-lfs to track datasets. Users typically use a single repository per pipeline, but can use the same repository to version multiple pipelines as well. With git-lfs, instead of writing large blobs to a repository it writes small pointer files that contains the hash of the original file, the size of the file, and the version of

git-lfs used. The files themselves are stored separately which makes the size of the repository small and manageable with git. The main reason why we chose git and git-lfs for version control is that git is the de facto standard for versioning source code, and we want to include versioning of datasets without altering the typical development workflow.

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

3.2.4 Pipeline Reconfiguration and Re-execution

Reconfiguring a pipeline is common practice in precision medicine, e.g. if we later would like to change tool parameters or add additional steps to the analysis pipeline. To reconfigure an existing pipeline users make the applicable changes to the pipeline description and re-run it using walrus. walrus will then recompute the necessary steps and return a version ID for the newly run pipeline. This ID can be used to compare pipeline runs, the changes made, and optionally restore the data and configuration from a previous run. Reconfiguring the pipeline to use updated tools or reference genomes will alter the pipeline configuration and force walrus to recompute the applicable pipeline stages.

The command-line interface of walrus provides functionality to restore results from a previous run, as well as printing information about a completed pipeline. To restore a previous pipeline run, users use the `restore` command 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.

3.3 Evaluation

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

3.3.1 Clinical Application

We have used walrus to analyze a whole-exome dataset to from a sample in the McGill Genome Quebec [MGGQ] dataset (GSE58644)[10] to discover SNPs, genomic variants and somatic mutations. We interactively developed a pipeline description that follows the best-practices of The Broad Institute[8] and generated reports that summarized the findings to share the results.

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

3.3.2 Performance and Resource Usage

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 [25] using tools from picard and the GATK. We show the storage and computational overhead of our approach, and the benefit of capturing the pipeline specification using a pipeline manager rather than a methods section in a paper. The pipeline description, data, and code is available along with walrus at `github.com/uit-bdps/walrus`. We use the popular NA12878 dataset since it makes it possible to easily share both the pipeline description and resulting datasets.

We first run the variant calling pipeline without any additional provenance

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

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 show the runtime and storage overhead of introducing this pipeline change. 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.

Table 3.1 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. 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 and git only has to modify the pointers to the correct files in the `.git` subdirectory.

Table 3.1: Runtime and storage use of a 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

3.4 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 Graphical User Interfaces (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.[26] A pipeline is written as a JSON or YAML file, or a mix of the two, and describes each step in detail, e.g. what tool to run, its input parameters, input data and output data. The pipeline descriptions are text files that can be version controlled and shared between projects. There are multiple implementations of CWL workflow platforms, e.g. the reference implementation cwl_runner,[9] Arvados,[3] Rabix,[4] Toil,[5] Galaxy,[1] and AWE.[6] 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.[27] It is possible to run Galaxy on local compute clusters, in the cloud, or using the online Galaxy site[10] In Galaxy users set up an analysis pipeline using a web-based graphical interface, and it is also possible to export or import an existing workflow to an Extensible Markup Language (XML) file.[11] We chose not to use Galaxy because of missing command-line and scripting support, along with little support for running workflows with different configurations.[2] Rabix provides checksums of output data to verify it against the actual output from the pipeline. This is similar to the checksums found in the git-lfs pointer files, but they do not store the original files for later. Arvados stores the data in a distributed storage system, Keep, that provides both storage and versioning of data. We chose not to use CWL and its implementations because of its relaxed restrictions on having to use containers, its verbose pipeline descriptions, and the complex compute architecture required for some of the runners. We are however experimenting with an extension to walrus that translates pipeline descriptions written in walrus to CWL pipeline descriptions.

Pachyderm is a system for running big data analysis pipelines. It provides complete version control for data and leverages the container ecosystem to provide reproducible data processing.[12] Pachyderm consists of a file system

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

(Pachyderm File System (PFS)) and a processing system (Pachyderm Processing System (PPS)). PFS is a file system with git-like semantics for storing data used in data analysis pipelines. Pachyderm ensures complete analysis reproducibility by providing version control for datasets in addition to the containerized execution environments. Both PFS and PPS is implemented on top of Kubernetes.[13] We believe that the approach in Pachyderm with version controlling datasets and containerizing each pipeline step is the correct approach to truly reproducible data analysis pipelines. The reason we did not use Kubernetes and Pachyderm was that it was not available on the required in-house compute infrastructure.

As discussed in [28], recent projects propose to use containers for life science research. The BioContainers[24] and Bioboxes[29] projects addresses the challenge of installing bioinformatics data analysis tools by maintaining a repository of docker containers for commonly used data analysis tools. Docker containers are shown to have better than, or equal performance as VMs.[30] Both forms of virtualization techniques introduce overhead in I/O-intensive workloads, especially in VMs, but introduce negligible CPU and memory overhead. For genomics pipelines the overhead of Docker containers will be negligible since these tend to be compute intensive and they typically run for several hours [30] 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.[31]

3.5 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 discover e.g. genomic variants, facilitating simple sharing of pipeline descriptions, or simplifying reproducing analysis results are still areas that need improvement. Pipelines typically need to be tailored to fit each project and patient, and different patients will typically elicit different molecular patterns that require individual investigation. While we could follow best practices to develop our pipeline we explored different tools and parameters before we arrived at the final analysis pipeline. For example, in our Whole-exome sequencing (WES) pipeline we ran several rounds of preprocessing (trimming reads and quality control) before we were sure that the data was ready for analysis. Having a pipeline system that could keep track of different intermediate datasets, along with the pipeline specification, simplifies the task of comparing the results from pipeline tools and input parameters. While we

13. kubernetes.io

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. While git-lfs supports large files it still takes a significant amount of time to add large files to a repository. This makes the entire analysis pipeline slower, but we argue that having the files version controlled outweigh the runtime. In addition there are as time of writing few public hosts that allow uploads of datasets larger than a few gigabytes. This makes it necessary to host git-lfs servers on-site and makes sharing of full analysis repositories cumbersome. We hope that in the future git-lfs can provide a standard way of sharing large datasets in genomics.

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 formally investigated 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.

3.6 Conclusions

We have designed and implemented walrus, a tool for building 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 required features to be used in a clinical setting. We have used walrus to analyze a patient's metastatic lesions and adjacent normal tissue to provide insights and recommendations for the cancer treatment.

4

Interactive Exploration

In this chapter we outline our approach for building interactive data exploration applications. We demonstrate the usefulness of the approach through a set of different applications for exploring transcriptional profiles from the NOWAC cohort. While these are specialized applications, we show how the design patterns and ideas can be used in a wide range of disciplines. We have developed the approach from implementing a set of applications, and outline how our experience has shaped it.

4.1 Data and Tools

In recent years the biological community has generated an unprecedented amount of data. While the cost of data collection has drastically decreased, data analysis continue to represent a large fraction of the total cost of these studies.[**?**] Data analysis tools, especially those designed specifically for the project at hand, provide clear benefit to the human experts who are interpreting data and deriving results.

Often in systems biology studies, the ability to explore newly generated data relative to prior knowledge located in third-party databases and software systems is key. This includes, for example, entities such as the Gene Ontology (GO),[1]

1. `geneontology.org`.

the Kyoto Encyclopedia of Genes and Genomes (KEGG),[2] and the Molecular Signatures Database (MSigDB)[3] that together catalog the function of nearly every gene, gene product, pathway or cellular process. These tools, and most bioinformatics databases in general, offer interfaces for data retrieval.

Visual explorative analysis is essential for understanding biological functions in large-scale omics' datasets. However, enabling the inclusion of omics' data in large epidemiological studies requires collecting samples from thousands of people at different biological levels over a long period of time. It is therefore usual to reuse the data for different research questions and projects. Although an existing tool may be useful for one project, no tool provides the required functionality for several different projects.

Data analysis in systems biology is greatly reliant on programming languages especially tailored to these domains, providing easy direct access to specific algorithms and statistical routines. The R statistical programming language provides developers open access to thousands of libraries through repositories such as CRAN[4] or Bioconductor[5]. Similarly, other languages such as Python and Go have bioinformatic extensions including BioPython[**?**] and biogo[**?**] respectively, providing domain specific routines. Although tremendously helpful, different tools and languages are used in different domains of systems biology for many reasons. This creates a need for novel approaches to integrate the different libraries between the programming languages and tools.

4.2 Motivating Examples

4.2.1 Kvik Pathways

The aim of the first application was a to simplify the exploration of the results from a previous published project ([8], doi: 10.1371/journal.pone.0067270) that compared gene expression in blood from healthy women with high and low plasma ratios of essential fatty acids. Gene expression differences where assessed and determined that there were 184 differentially expressed genes. When exploring this list of 184 genes, functional information was retrieved from GeneCards and other repositories, and the list was analyzed for overlap with known pathways using MSigDB (available online at broadinstitute.org/gsea/msigdb). The researchers had to manually maintain overview of

2. `kegg.jp`.
3. `software.broadinstitute.org/gsea/msigdb`.
4. `cran.r-project.org`.
5. `bioconductor.org`.

single genes, gene networks or pathways, and gather functional information gene by gene while assessing differences in gene expression levels. With this approach, researchers are limited by manual capacity, and the results may be prone to researcher bias.

4.2.2 Matched Interactions Across Tissues (MIxT)

The aim of the Matched Interactions Across Tissues (MIxT) study was to identify genes and pathways in the primary breast tumor that are tightly linked to genes and pathways in the patient blood cells.[**?**] We generated and analyzed expression profiles from blood and matched tumor cells in 173 breast cancer patients included in the Norwegian Women and Cancer (NOWAC) study. The MIxT analysis starts by identifying sets of genes tightly co-expressed across all patients in each tissue. Each group of genes or modules were annotated based on a priori biological knowledge about gene functionality. Focus was placed on 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).

We built an R package, mixtR,[6] with the statistical methods and static visualizations for identifying associations between modules across tissues. To make the results more easily accessible we built a web application that interfaces with the R package, but also online databases to retrieve relevant metadata. To make it possible to easily update or re-implement parts of the system without effecting the entire application, it was developed 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.

4.3 Experience and observations

From our experience in developing a framework for building data exploration applications, we identified four requirements such applications should satisfy:

Interactive The applications should provide interactive exploration of datasets through visualizations and integration with relevant information. To understand the large quantities of heterogeneous data in epidemiological studies, researchers need interactive visualizations that provide different views and presentations of the data. Also, to understand the results it

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

is important to have instant access to existing knowledge from online databases.

Familiar They 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.

From our experience 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 into a data exploration application.

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

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

Our experience can be generalized 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.

Microservice Design and Implementation

In the rest of the section we describe how we designed and implemented two microservices in Kvik[11] which we later used to build the MIxT web application.

Compute Service

The main goal of a data exploration application in systems biology 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. Because these tools are built to provide specialized analyses, they often don't provide a reusable interface outside the programming environment they are built in.

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

To alleviate application developers of these challenges, 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,[7] MSigDB, HGNC, and KEGG.

Both the compute and the databases service in Kvik build on the standard *net/http* package in the Go programming language.[8] The database service use the *gocache*[9] package to cache any query to an online database. In addition we deploy each service as Docker containers.[10]

4.4 Applications

4.4.1 Matched Interactions Across Tissues (MIxT)

We show the viability of the microservices approach in Kvik by describing the MIxT web application for exploring and comparing transcriptional profiles from blood and tumor samples. We conduct an initial evaluation to illustrate that we can built interactive applications using the microservices provided by Kvik.

7. `eutils.ncbi.nlm.nih.gov`.
8. `golang.org`
9. `github.com/fjukstad/gocache`.
10. Available at `hub.docker.com/r/fjukstad/kvik-r` and `hub.docker.com/r/fjukstad/db`.

Analysis Tasks

The web application provides functionality to perform six data analysis tasks (A1-A6):

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.

Design and Implementation

From these 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 4.1 shows the system architecture of MIxT which consists of three parts i) the web application itself containing the user-interface and visualizations; ii) the compute service performing the MIxT analyses developed in an R package, delivering data to the web application; and iii) the database service providing up-to-date information from biological databases. Each of these components run within Docker containers making the process of deploying the application simple.

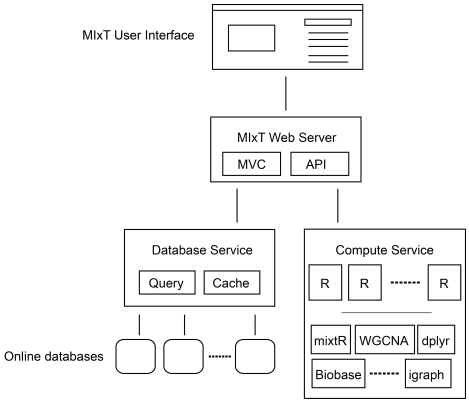


Figure 4.1: The architecture of the MIxT system. It consists of a web application, the hosting web server, a database service for retrieving metadata and a compute service for performing statistical analysis. Note that only the web application and the R package are specific to MIxT, the rest of the components can be reused in other applications.

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 4.2 shows the web page presented to users when they access the co-expression gene set 'darkturquoise' from blood. To explore the co-expression relationship between genes (**A2**) we use an interactive graph visualization build with Sigmajs[11]. 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 using the d3[12] library. We built a simple upload page where users can specify their gene sets (**A5**). The file is

11. `sigmajs.org`.
12. `d3js.org`.

uploaded to the web application which redirects it to the compute service that runs the analyses. Similarly we can take user input to search for genes and processes (**A6**).

Figure 4.2: 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.

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.

4.4.2 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. **??** 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.

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

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

Initially, Kvik Pathways was implemented to explore gene expression data from a not yet published dataset. To use Kvik Pathways to explore the data from the analyses in [8], we only needed to make small modifications to the analysis R script used by the Data Engine. (The modified R script is found at github.com/fjukstad/kvik/dataengine/data-engine.r). Instead of loading the unpublished dataset, we could load the dataset from [8] and use the four functions that are accessible over RPC (Table 1 shows the HTTP API which uses the underlying RPCs). Currently this script is less than 30 lines, consisting of four functions to retrieve data and a simple initialization step that reads the dataset. Researchers only have to modify these four functions to enable exploration of new datasets. As of the current implementation of Kvik Pathways researchers have to modify the analysis script outside the application.

Use Case

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 [8]). 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.

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.

4.4.3 Other Disciplines

We have also used the microservice architecture in an application where users can upload and explore air pollution data from Northern Norway.[32] 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.

4.5 Evaluation

To investigate if it is feasible to implement parts of an application as separate services, we evaluate the response times for a set of queries to each of the two supporting services.

To evaluate the database service we measure the query time for retrieving

information about a specific gene with and without caching.[13] This illustrates how we can improve performance in an application by using a database service rather than accessing the database directly. We use a AWS EC2 *t2.micro*[14] instance to host and evaluate the database service. The results in Table 4.1 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 4.1: 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.[15] 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 4.2 shows the time to complete the benchmark for different number of concurrent connections. We see that the compute service in Kvik performs better than the OpenCPU[16] alternative. We believe that speedup is because we keep a pool of R processes that handle requests. In OpenCPU a new R process is forked upon every request that results in any computation executed in R. Other requests such as retrieving previous results do not fork new R processes.

Table 4.2: 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

13. More details online at `github.com/fjukstad/kvik/tree/master/db/benchmark`.
14. See `aws.amazon.com/ec2/instance-types` for more information about AWS EC2 instance types.
15. More details at `github.com/fjukstad/kvik/tree/master/r/benchmarks`.
16. Built using the *opencpu-server* Docker image.

4.6 Related Work

OpenCPU is a system for embedded scientific computing and reproducible research.[**?**] 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.[17]. The compute service in Kvik follows many of the design patterns in OpenCPU. Both systems interface with R packages using a hybrid state pattern over HTTP. Both systems provide the same interface to execute analyses and retrieve results. Because of the similarities in the interface to R in Kvik we provide packages for interfacing with our own R server or OpenCPU R servers.

Shiny is a web application framework for R[18] 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.[**?**] 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.

Visualization

Cytoscape is an open source software platform for visualizing complex networks and integrating these with any type of attribute data.[**?**] Through a Cytoscape App, cyREST, it allows external network creation and analysis through a REST API[**?**], making it possible to use Cytoscape as a service. To bring the visualization and analysis capabilities to the web applications the creators of Cytoscape have developed Cytoscape.js[19], 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 bioinfor-

17. `opencpu.org/apps.html`.
18. `shiny.rstudio.com`.
19. `js.cytoscapejs.org`.

matics community.[**?**] BioJS builds on node.js[20] 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.

20. `nodejs.org`.

5

Conclusion

A

Publications

Bibliography

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

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

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

[4] 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.

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

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

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

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

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

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

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

[19] N. R. Council *et al.*, *Toward precision medicine: building a knowledge network for biomedical research and a new taxonomy of disease*. National Academies Press, 2011.

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

[21] 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.

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

[23] 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.

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

[25] 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.

[26] 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.

[27] 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.

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

[29] 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.

[30] 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.

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

[32] 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.