Development of an Interface for Extending a Machine Learning Platform

A Design Science Research Study

Oskar Lund & Jørgen Skimmeland

Thesis submitted for the degree of
Master in Informatics: Programming and system architecture
60 credits

Institute for Informatics
Faculty of Mathematics and Natural Sciences

UNIVERSITY OF OSLO

Autumn 2023

Development of an Interface for Extending a Machine Learning Platform

A Design Science Research Study

Oskar Lund & Jørgen Skimmeland

Abstract

Background

Extensibility is an important aspect of software platforms, as it allows for integrating external tools and packages. This is particularly relevant in the field of bioinformatics, where the analysis of large datasets often requires the use of specialized tools. The immuneML platform, an open-source platform for analyzing adaptive immune receptor repertoires, currently relies on contributions to its source code via its GitHub repository.

Method

In this thesis, we follow the Design Science Research methodology, developing an interface that enables the integration of external tools with the immuneML platform. We conducted a literature review and designed evaluations that followed established scientific practices. We evaluated the interface's design through a focus group, a semi-structured interview, a quasi-experiment, and a survey with tasks.

Result

The results of our research show that an interface can improve the extensibility of a machine-learning platform. We demonstrate how the interface enables the extension of the platform without the need to work directly with the source code. Additionally, the interface facilitates the integration of external tools and enables the use of tools written in different programming languages.

Conclusion

Our research contributes to the field of software engineering by presenting an interface that increases the extensibility of the immuneML platform, allowing for the integration of external tools. Through two evaluations, we demonstrated the usability and effectiveness of the interface in facilitating external tool integration. Our findings highlight the potential benefits of using our interface in a machine-learning platform used within the field of bioinformatics and suggestions for further work.

Acknowledgements

I want to express my gratitude to our supervisors, Antonio Martini and Karthik Shivashankar, for their guidance, support, and valuable insights throughout the thesis project. Their encouragement, expertise, and commitment have been invaluable to completing this work. I am also grateful to the immuneML team for their cooperation. Special thanks to Jørgen for the good collaboration.

Finally, I want to thank my friends and family for their support and encouragement throughout writing this thesis.

Oskar Lund

University of Oslo

May 2023

First and foremost, I would like to express my sincere gratitude to our supervisors, Antonio Martini and Karthik Shivashankar, for their academic guidance, encouragement, and support.

To my family I want to express my deepest gratitude to my parents, Rita and Lars, and my sisters, Hanna and Sunniva, for their support, encouragement, and belief in me throughout my six years of studies. Additionally, I would like to thank all my friends for their encouragement and motivation.

At last, I would like to thank Oskar for being such a great working partner, making the experience of writing this thesis a good experience.

Jørgen Skimmeland

University of Oslo

May 2023

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Extensibility is an essential aspect of software development, allowing software to adapt to constantly changing needs and allowing new technologies to be adapted. According to Henttonen et al. [18], extensibility involves extending software with new components and features without compromising functionality or requirements. This property is essential for enabling software evolution and significant software reuse [57] and is considered a key quality of software development, especially in open-source software [18].

Extending software platforms can bring challenges, especially in cases where developers must work directly with source code to make contributions. This approach can limit the number of developers that are willing and able to contribute to such a platform. In the case of immuneML, a machine-learning platform used in bioinformatics, the current state of extensibility requires developers to add code directly into the platform's source code, making the process challenging. This thesis addresses this problem by proposing a novel approach to improving the extensibility of immuneML. This is done by developing an interface that enables tools to be developed and run outside of the immuneML's software core with the aim of keeping the process simple to reduce the barrier for developers to extend the platform with new tools. This formulates our overall research question:

> *RQ1: How can extending a machine learning platform be improved through an interface?*

Improvement is an abstract concept. While creating a new way of extending immuneML itself by introducing an interface can bring improvements, this alone does not specify what this improvement is. Therefore, we introduce the concept of simplicity. Following a more technical description in the context of software development, simplicity can be defined as *"The degree to which a system component has a design and implementation that is straightforward and easy to understand"* [21]. Applying this definition to the interface, which acts as a mediator between immuneML and external tools, examining its simplicity can provide insights into whether the interface has improved the platform's extensibility. Consequently, we pose a new sub-research question:

> *RQ1.1: How can extending a machine learning platform be simplified through an interface?*

The field of bioinformatics includes numerous tools written in various programming languages. Currently, immuneML is restricted to utilizing only Python code for extending its platform, with no support for other programming languages. Although the addition of an interface enabling external tools to operate outside the platform's software core is a step in expanding the platform's extensibility, limitations persist. Therefore, as another improvement area of extensibility, our study aims to enable the use of tools in other programming languages to overcome this limitation. This brings us to the last research question:

> *RQ1.2: How can an interface facilitate the integration of tools written in additional programming languages?*

Our study follows the Design Science Research methodology, a problem-solving paradigm to enhance design knowledge by creating innovative artifacts [9]. In this study, an artifact was designed and developed to improve the extensibility of the immuneML platform. The developed artifact encapsulates the complexity of immuneML, enabling developers to add external code and extend the platform's functionality. Furthermore, the artifact introduces a previously nonexistent functionality where developers can extend immuneML with tools written in other programming languages, enhancing the platform's interoperability.

1.1 Structure

Chapter 2: Background

In the background chapter, we introduce the concept of software extensibility and integration. We give a context into the platform of immuneML and present relevant research.

Chapter 3: Methodology

In the methodology chapter, we present the research methodology design science research. We give an overview of the different research strategies and methods used.

Chapter 4: Iteration 1

We start this chapter by introducing the identification and motivation for the project. Following this, we present the work done for each activity in the first iteration.

Chapter 5: Iteration 2

This chapter presents the work done for each activity belonging to the second iteration, resulting in the final artifact.

Chapter 6: Final Artifact

In this chapter, we present the final artifact. The general solution, architecture, and functionality are explained in detail.

Chapter 7: Discussion

The discussion chapter offers reflections on the research and our findings. We discuss the research's contributions, limitations, research quality, practical and theoretical implications, and suggestions for further work.

Chapter 8: Conclusion

In our final chapter, we conclude our work and present recommendations for future research.

Chapter 2

Background

In this chapter, we provide an overview of the relevant concepts for our thesis, to give the readers a contextual foundation for understanding our project. We first introduce the immuneML platform and provide an overall context of our project. Next, we present the concept of software architecture and extensibility, which underpins the development of our project. We briefly introduce the diversity of programming languages in bioinformatics. Finally, we briefly introduce machine learning, the technologies used in the thesis, and the software used for our use cases.

2.1 Context

2.1.1 immuneML

In this section, we provide context for immuneML, an open-source platform for research on the immune system in bioinformatics [34]. The platform is specifically designed for analyzing adaptive immune receptor repertoires (AIIR) using machine learning (ML).

The platform implements all steps of the AIRR ML process, from preprocessing to model training and interpretation [34]. It is designed to be user-friendly and accessible to researchers who may not have knowledge in programming or ML. The platform can be used locally, using a command-line interface, and online, through a web interface. In this project, we focus on running the platform locally on a computer, where the user defines which analysis should be performed by a specification file, YAML. The structure of the YAML file can be seen in Appendix H.

Briefly listed, the immuneML platform can be used for [51]:

- training ML models to predict diseases and antigen bindings
- exploratory analysis of datasets

- stimulating immune events into synthetic or experimental repertoire datasets
- applying trained ML models to datasets that has unknown class labels

Extensibility

immuneML's extensibility is a key feature that enables users to adapt the platform to their research needs [34]. The platform's source code is available on GitHub, and the platform developers encourage contributions from the community. immuneML has a modular architecture that can be extended through glass-box extensibility mechanisms, defined in Section 2.3.1. These mechanisms allow users to implement new functionalities, such as preprocessing, encodings, and ML methods, by using abstract classes. Abstract classes enable users to extend immuneML's functionalities while ensuring that the new code integrates with the existing framework. However, this solution faces certain challenges: Contributors must work directly with the source code to extend it, and the size of the platform increases with each contribution.

Conceptualizing immuneML as a Platform

As Reuver, Sørensen and Basole [38] states, digital platforms are a challenging research object due to their distributed nature and intertwinement with technologies, markets, and institutions. The definition of a platform depends on the perspective of the different fields, highlighting the importance of specifying the definition for the context.

When referring to immuneML as a platform in this thesis, we follow the description by Ghazawneh and Henfridsson [15]; a digital platform is an extensible codebase of a software-based system that provides shared core functionality for the different modules that interoperate with it. These modules communicate with each other through interfaces provided by the platform. We follow this description based on immuneML being an open-source codebase available on GitHub that allows for external contributions.

2.2 Software architecture

In this section, we introduce software architecture, quality attributes, and define the concept of an interface.

The software architecture of a system can be defined as *"the set of structures needed to reason about the system. These structures comprise software elements, relations among them, and properties of both"* [4]. The architecture is an abstraction of the system, and only the structures that support reasoning about the system and its properties should be described [4].

The software architecture should be abstract to a level where non-technical stakeholders can understand it to a necessary extent and, at the same time, maintain an abstraction that can be refined into a sufficient level of technical specification that can be used to guide implementation, integration, testing, and deployment [4]. The architecture should, in other words, provide a common language between the technical and non-technical stakeholders.

2.2.1 Quality Attributes

Bass, Clements and Kazman [4] defines a quality attribute as *"a measurable or testable property of a system that is used to indicate how well the system satisfies the needs of its stakeholders beyond the basic function of the system."* There are several quality attributes. However, we chose three that we found most fit to focus on for the design and development of the artifact: usability, maintainability, and interoperability.

Usability

Usability refers to how easy it is for a user to accomplish a desired task and what kind of user support the system provides [4]. It comprises areas such as using a system efficiently, learning system features, minimizing the impact of user errors, and increasing confidence and satisfaction.

The reason why we chose usability as a quality attribute is that we wanted to make an interface simple to use, facilitating a simplified process of extending immuneML.

Maintainability

Maintainability can bed defined as *"The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment"* [21].

Maintainability was focused on making sure the interface would be easy to modify and that it could accommodate future changes in the immuneML platform.

Interoperability

Interoperability refers to the degree to which systems can usefully exchange meaningful information. [5] For the information to be meaningful for the systems, they must have the ability to exchange the data (syntactic interoperability) and also be able to interpret the data correctly (semantic interoperability) [5].

The artifact's importance lies in its ability to facilitate communication between immuneML and external tools. Thus, we looked at interoperability as one of the key attributes in the design and development.

2.2.2 Interface

Bass, Clements and Kazman [4] defines an interface as *"... boundary across which elements meet and interact, communicate, and coordinate."* The interface for a software element controls which internal functionality and properties are available for the outside.

Interfaces are responsible for establishing a contractual specification, allowing elements to collaborate and exchange information [4]. One of the motivations for creating and using interfaces is how they encapsulate software [4]. The encapsulation makes software free to evolve without impacting the elements that use the interface. In other words, only the changes to the interface impact these elements and not the software itself.

2.3 Extensibility

Extensibility refers to extending software with new components and features without losing functionality or qualities specified as requirements [18]. The requirement for a software/system to be considered extensible is that it can adapt to possible unanticipated changes in the software specification [57]. Extensibility boosts the reuse of software significantly and is an essential property for enabling the evolution of software [57].

Henttonen et al. [18] writes that some of the most important qualities of software development are integrability and extensibility. In open-source software, the importance of integrability and extensibility is paramount. By integrability, we refer to an ability to make components of a system separately developed to work correctly together [18].

2.3.1 Extensibility mechanisms

Software extensibility can be explained in different forms. Zenger [57] proposes a classification of extensibility mechanisms based on what artifacts are changed and in what way these artifacts are changed. These are broadly separated into white-box, gray-box, and black-box extensibility.

White-Box Extensibility refers to what ways software systems can be extended by directly modifying or adding to the source code [57]. This extensibility mechanism is further divided into open-box and glass-box extensibility.

> **Open-box extensibility** refers to extensible systems where changes are inserted directly into the source code [57]. Prerequisites are that the source code must be available; the process of changing the source code is considered an error-prone and tedious activity and does not preserve backward compatibility.

> **Glass-box extensibility**, on the other hand, refers to how software systems with source code available can be extended without directly modifying it [57]. The developers can view the source code, but to extend it, they have to separate their code to not affect the original system. In projects using object-oriented programming languages, like Python, this can be achieved by inheriting from base classes.

Black-Box Extensibility refers to how software can be extended without having internal details about its architecture and implementation [57]. This enables the encapsulation of the software; if wanted, it can hide the source code from others. Black-box extensibility is often more limited than white-box; however, the important point is that they generally are easier to extend because they require less knowledge of a system's internal structure Zenger.

Gray-Box Extensibility refers to an approach for extending software without having to rely on the availability of the source code [57]. It is considered a compromise between white- and black-box extensibility in the sense that it does not depend on total exposure to the source code but, at the same time, provides details about a system and how to extend it.

2.4 Diversity of programming languages in Bioinformatics

Bioinformatics is considered an interdisciplinary field involving biology and genetics, mathematics, statistics, and computer science [10], and is in modern biology one of the major areas of study [30]. The field addresses data-intensive and large-scale biological problems from a computational viewpoint.

Within the bioinformatics domain, there are thousands of applications (software) used for anything from analyzing DNA sequences to optimizing and predicting organism growth [11]. Similarly to the broader software development field, software in bioinformatics is developed in a wide range of programming languages. This is a result of programming language design which is based on trade-offs such as convenience, strictness, and performance [6].

In bioinformatics, the software is typically written in Java, C, C++, Python, Perl, R, and Ruby, with Python and Perl being particularly popular [6]. As Grimmer et al. [17] notes, there is no programming language that is best for solving all kinds of problems. Java is an example of a general-purpose programming language that can be used in many situations [42]. However, it does not fit every job. When dealing with statistics, R is commonly preferred, while in the context of speed and performance, C++ is more appropriate.

The diversity of software written in programming languages introduces an important discussion. Bonnal et al. [6] points out a question developers have to make when faced with a functionality that is not currently in a project but exists in a different programming language. The developers must then ask themselves, should they rewrite the code, essentially duplicating it, or create a bridge from one language to another, allowing for the use of functionality without having to go through the effort of rewriting?

While there is a wide range of programming languages, Bonnal et al. [6] states that bioinformaticians cannot be expected to become sufficient in every programming language, additionally, writing the same functionality across several programming languages is inefficient.

2.5 Machine learning

Zhou [59], defines ML as *"the technique that improves system performance by learning from experience via computational methods."* In computer systems, this experience is represented by data, such as numeric tabular data or images. The main task of machine learning is to develop learning algorithms that build models from this data. This process is called training, and the result of the training is a model that can be used to make predictions on new observations.

2.6 Technologies

Throughout our DSR study, we have used technologies and software to shape and develop the artifact. In this section, we briefly introduce these so it will be easier to follow when reading about our design and development of the artifact.

We also present the two programs (tools) Absolut and DeepRC, used in our use cases to demonstrate the effectiveness of our artifact. These use cases are introduced in Chapter 4, and presented in Chapter 5.

2.6.1 Inter-Process Communication

Interprocess communication (IPC) can be defined as a mechanism that allows processes in an operating system to communicate with each other [33]. Put in other words, IPC mechanisms are used for transferring data between processes [50]. The processes can be either in the same system or different systems. While IPC enables communication, it also involves synchronization of the actions performed by processes and managing the data-sharing activity [33].

There are several types of IPC mechanisms, some of them being: pipes, Message Queues, Semaphores, Shared Memory, and Sockets [33]. In our artifact, sockets are essential to enable communication between immuneML (one process) and the external tool (another process). Sockets are described as a bi-directional communication mechanism [50].

2.6.2 ZeroMQ

ZeroMQ is a universal messaging library [58]. The messaging library is a concurrency framework, providing sockets that can be used to carry atomic messages through various transport mechanisms such as inter-process, TCP and multicast [58]. One important aspect of the library is its focus on facilitating code that is clean, modular, and easier to scale [14]. Low-level details of socket types, routing, and connection handling framing can be complicated, and this is what ZeroMQ attempts to solve [16].

ZeroMQ hides complexity in applications by providing a layer of abstraction on top of the traditional socket API [16]. More technically explained, as opposed to being stream (TCP) or datagram (UDP) oriented, the communication in ZeroMQ is message-oriented. This means that we do not have to implement any explicit buffering or framing. As a result of ZeroMQ handling all framing and buffering, the client and server applications become both simpler, more secure, and easier to write [16].

Besides the technical description of ZeroMQ, another important aspect of ZeroMQ is its support for popular programming languages such as C, C++, Java, Python, and Go. The messaging library facilitates a relatively easy pattern for implementation no matter what language you use (that the library supports). It also runs on most operating systems, leading to increased accessibility.

2.6.3 Software for use cases

Absolut!

Absolut!, developed by Greifflab, is a software used for generating unconstrained lattice antibody-antigen bindings [2]. It consists of a database and a user interface written in the programming language C++, enabling the custom generation of custom antibody-antigen structural datasets that can be used for machine-learning training and testing [1].

DeepRC

DeepRC is a machine learning method for immune repertoire classification [53]. The method *"...integrates transformer-like attention, or equivalently modern Hopfield networks, into deep learning architectures for massive MIL."* [53] In our use case, we are using a Python project [31] that implements the DeepRC method.

Chapter 3

Methodology

In the methodology chapter, we present an overview of the research methodology Design Science Research (DSR). Firstly, we will introduce the concept of DSR and explain the overall process we followed. For each activity of the process, we will be giving a theoretical introduction and explaining our approach to each one, with weight on the evaluation activity. Furthermore, we will discuss the data collection methods we used and explain how we conducted our data analysis, including both methods for quantitative and qualitative analysis.

3.1 Design Science Research

Design Science Research (DSR) is a problem-solving paradigm to enhance design knowledge by creating innovative artifacts [9]. The research method aims to produce innovative solutions that solve real-world problems, producing knowledge of how things can and/or should be designed, which provides a fuller understanding and enhances human knowledge [9]. Specifically, the desire is to improve environments by introducing new and innovative artifacts and the process used to build them [19].

The artifacts can be represented by models, methods, constructs, and instantiations [20]. We consider the final artifact developed in this project as an instantiation. An instantiation artifact refers to a working system usable in practice [25], which is a software component in our case.

3.2 Process description

There are several process models used for DSR projects. According to Brocke, Hevner and Maedche [9], one of the most referenced models is the DSR process model proposed by Peffers et al. [35]. By analyzing previous research, the authors developed the DSR methodology model by building upon the experiences of other researchers in the field [35]. They aimed to establish a generally agreed-upon framework for research based on Design

Science principles. The methodology was designed using a consensus-building approach, focusing on shared understanding rather than subtle differences in views among researchers [35].

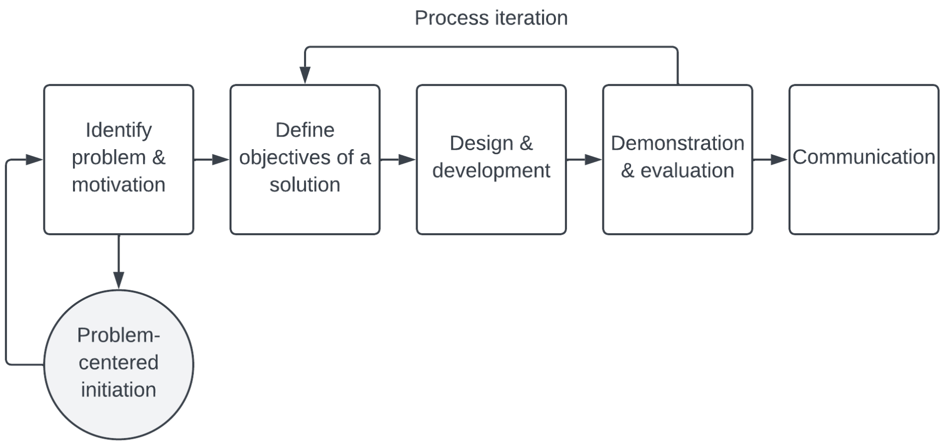

Process iteration
Identify problem & motivation
Define objectives of a solution
Design & development
Demonstration & evaluation
Communication
Problem-centered initiation

Figure 3.1: Our DSR process model adapted from Brocke, Hevner and Maedche [9]

In our research, we adapted the process described by Brocke, Hevner and Maedche [9]. Figure 3.1 depicts how the process was applied in our project, where we performed two process iterations, as described in Chapter 4 and Chapter 5. The figure shows five activities:

1. problem identification and motivation
2. defining the objectives for a solution
3. design and development
4. demonstration and evaluation
5. communication

The original model by Brocke, Hevner and Maedche [9] depicts demonstration and evaluation as two separate activities. However, these two are tightly coupled, and these have been done simultaneously in our project. In the model, activities 2, 3, and 4 describe one single iteration, beginning with defining objectives and ending with demonstration and evaluation. The original model by Brocke, Hevner and Maedche [9] also shows how to choose whether the next iteration starts with defining objectives or design and development. However, each of our iterations started with defining our objectives for the iteration.

Additionally, the model describes entry points, which can be used to identify which activity one should have as a starting point. Our project had a problem-centered approach, which is the basis of the nominal sequence, starting with activity 1. We followed this sequence because our idea for the research resulted from an observation of the problem presented by the immuneML team.

3.3 Problem identification and motivation

The first activity described in the process model is the problem identification and motivation activity. This activity involves defining the specific research problem and why it is valuable to find a solution [9]. This serves two purposes: 1. it inspires the researcher and the research audience to seek a solution, and 2. it demonstrates the researcher's understanding of the problem.

To identify the problem and find motivation, we had an initial discussion with the immuneML team early on in the project. They presented potential areas for improvement in the immuneML platform that could be explored from a software engineering perspective. Based on this discussion, we conducted a literature review to explore literature related to the problem area, which we presented in Chapter 2.

3.4 Objectives of solution

The second activity in the process model is defining the objectives for a solution. The problem definition and knowledge of what is both possible and feasible can be used to derive the objectives of a solution [35].

The objectives for the first iteration were established based on the initial discussion with immuneML, as mentioned in Section 3.3, and our understanding of the problem. In our second iteration, we refined the objectives based on the evaluation result from the first iteration.

3.5 Design and development

In this activity, we design and develop the artifact. Essential parts of this activity involve determining the artifact's wanted functionality and its architecture before creating the artifact [35].

In the first iteration, described in Chapter 4, we explored potential solutions by creating prototypes and models. In the second iteration, described in 5, we developed the final artifact based on the models and prototypes produced in the first iteration and feedback from the evaluation with the immuneML developers.

3.6 Demonstration

In the demonstration activity, we demonstrate the use of the artifact for one or more of the problems it was initially designed to solve [35]. As referred to in Section 3.2, we merged the demonstration activity with the evaluation, but we briefly summarize it in this section.

During the demonstration in the first iteration, we presented initial prototypes and a proposed structure for our solution. We also showcased several models illustrating what a possible solution could look like.

In the second iteration, the artifact was demonstrated through a quasi-experiment and a survey with tasks where participants used it. Furthermore, models and a technical description of the solution were presented during a semi-structured interview as part of the evaluation process.

3.7 Evaluation

As Venable, Pries-Heje and Baskerville [48] describes, there is no science in Design Science Research without performing an evaluation. It is widely recognized as a central and essential activity for conducting rigorous Design Science Research [48].

According to Johannesson and Perjons, [25], the main goal of an evaluation is to discover to what extent an artifact effectively solves the problem for what it initially was designed. In addition, Johannesson and Perjons [25] explains that the goals of performing an evaluation can also be:

- investigating the knowledge about the artifact
- comparing the artifact to other already existing artifacts
- investigating the potential side effects of using an artifact

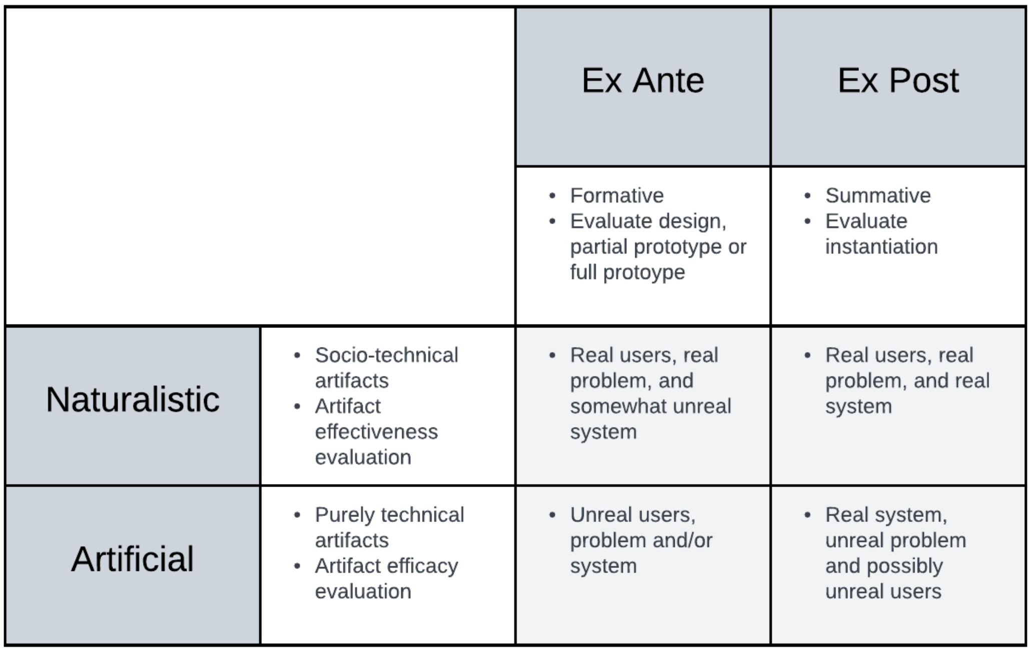
Our goal for performing artifact evaluation was to discover how well the developed artifact solved the problem it was designed for. In other words, we followed the primary goal of DSR evaluation as described by Johannesson and Perjons [25].

As an activity in the DSR process model, the evaluation involves comparing the objectives of an artifact to the observed results gathered from using the artifact in the demonstration activity [35]. This way, we can provide evidence on whether a designed artifact fulfills its intended purpose [48].

3.7.1 Framework for evaluation design

Designing the evaluation activity of a DSR project can be challenging due to a lack of guidance, with few specific guidelines available beyond suggestions of evaluation methods [48]. However, Venable, Pries-Heje and Baskerville [48] propose a DSR evaluation framework that extends a previous framework proposed by Pries-Heje, Baskerville and Venable [37].

The extended framework presents clear guidance on designing and performing evaluations in DSR projects and aims to help identify one or several DSR evaluation strategies [48]. Additionally, it aims to assist in choosing the appropriate evaluation method(s) for one or several evaluation strategies. To design the evaluation, we followed this framework, which will be described in Section 3.7.2.



		Ex Ante	Ex Post
		• Formative • Evaluate design, partial prototype or full protoype	• Summative • Evaluate instantiation
Naturalistic	• Socio-technical artifacts • Artifact effectiveness evaluation	• Real users, real problem, and somewhat unreal system	• Real users, real problem, and real system
Artificial	• Purely technical artifacts • Artifact efficacy evaluation	• Unreal users, problem and/or system	• Real system, unreal problem and possibly unreal users

Figure 3.2: Strategy selection adapted from Venable, Pries-Heje and Baskerville [48]. The figure shows the essential points extracted from the framework model that we used for our decision-making

Figure 3.2 shows an adapted version of the framework, presenting a model of four concepts used for the classification and characterization of different evaluation strategies [25]. These are ex ante vs. ex post evaluations and naturalistic vs. artificial evaluations.

Ex ante evaluations evaluate artifacts before they are designed, implemented, or constructed [49].

Ex post evaluation requires an artifact to be fully developed to be evaluated [25].

Naturalistic evaluations are conducted in the real world, where the artifact is used to solve real problems by real users [25]. Naturalistic evaluation allows a researcher to explore how a technology works in a real environment [47].

Artificial evaluations refer to an evaluation of an artifact in an unreal, artificial manner that does not reflect scenarios found in the real world [47]. It is conducted in an artificial setting, such as a laboratory [25].

In addition to the four dimensions, another often-used distinction is between **formative** and **summative** evaluations. Evaluating an artifact formatively refers to evaluating it while it is still under design [25]. The goal is to obtain information on how it can be improved during design activities. In contrast, a summative evaluation is conducted once an artifact has been designed and developed [25]. While the formative evaluation results are meant to feed back into the design process, the summative evaluation does not. Instead, the goal is to obtain a final assessment of the artifact's utility [25].

Selecting an evaluation strategy impacts the selection of the evaluation method used to evaluate an artifact. By using the framework, we can use the strategies to identify appropriate evaluation methods proposed by Venable, Pries-Heje and Baskerville [48].

3.7.2 Selecting evaluation strategy and method

The first step of using the extended framework is understanding the DSR evaluation context [48]. Using the model in Figure 3.2, we map the understanding to the criteria and select the strategies based on relevance. The second step is to use the selected strategy to identify the appropriate method(s). We will describe the details of these methods in Section 3.9.

Iteration 1: Strategy

Firstly, we had to identify what we wanted to achieve by performing the evaluation. In our first iteration, we did not have an implemented artifact, meaning that our evaluation had to be classified as an ex ante formative evaluation. Further, we had to decide between naturalistic and artificial evaluation. To create a proper discussion with inputs from different sides, we wanted to include both our supervisors and members of the immuneML team. This would be considered an evaluation in a naturalistic setting.

Iteration 1: Method

For the identified strategy, the framework presents focus groups as a suggested method. Focus groups effectively generate new ideas and encourage open and creative discussions among participants [25]. We found this method to be appropriate for this iteration because we wanted to explore a broader range of ideas.

Iteration 2: Strategy

In our second and final iteration, we developed a fully functional artifact, and our evaluation fell under the category of ex-post summative evaluation. To evaluate the use of the artifact, we intended to involve developers and users in testing our solution. However, we could not evaluate the entire development process using our artifact due to resource constraints, which meant that the evaluation had to be conducted in an artificial setting. Thus, this strategy would fall under the summative ex-post artificial evaluation strategy.

We also wanted to include the members of the immuneML team, where our idea was to present the artifact and get feedback on our solution. As this includes using real stakeholders, the real problem, and the finished artifact, this approach aligns with a naturalistic evaluation strategy.

In summary, we would follow two summative ex-post evaluation strategies: an artificial and a naturalistic evaluation.

Iteration 2: Method

Based on the artificial evaluation strategy, we saw that performing a quasi-experiment and a survey with tasks would be fitting methods for collecting data. By conducting a quasi-experiment, we can assess the impact of the final artifact on the user experience of the development process by comparing it to the previous solution. While not specifically listed as its own type of data collection method, we also saw it fitting to combine a task followed by a survey to evaluate how users experienced using our solution.

In our first iteration, we hosted a focus group to generate new ideas and discuss the artifact's design openly. In our summative evaluation, we wanted more specific data on our fully implemented artifact. We, therefore, saw it more fitting to conduct a semi-structured interview allowing both an open discussion and, at the same time, ensuring that specific data were collected.

3.8 Communication

The communication activity is the final step of the design science research process model. In this activity, essential information is communicated to relevant stakeholders. This includes discussing the problem and its importance, the artifact's functionality and originality, the thoroughness of its design, and its overall effectiveness [35]. The results of our conducted design science research are communicated through the presentation of this thesis.

3.9 Data collection

To gather data for our study, we used various data collection methods. Initially, we performed a literature review, which is outlined in Chapter 2. We hosted a focus group during our first iteration as part of our evaluation activity. We used a quasi-experiment, a survey with tasks, and a semi-structured interview during our final evaluation. In this section, we will explain how we used the methods for collecting data.

3.9.1 Literature review

Literature reviews are a way of gaining vital insight into scholarly topics, gathering published research on specific topics, surveying different research sources, and examining them critically [24].

Our literature review followed a less systematic and rigorous approach, meaning that we conducted a review that would be considered closer to a narrative literature review. Narrative literature reviews are discussions of important topics from a theoretical point of view and follow a less formal and rigorous approach than, for instance, a systematic approach [24]. The review does not involve reporting on methodology, search terms, and exclusion and inclusion criteria.

Our project began with a broad problem area focused on improving the immuneML platform from a software engineering perspective. Given the scope and complexity of this concept, we realized that a flexible and open-ended approach to our literature review would be most appropriate. Therefore, we opted for a narrative literature review. As we engaged in preliminary discussions with the immuneML team and conducted the literature review, we identified extensibility as our main focus. The content from our literature review is presented in Chapter 2.

3.9.2 Focus group

A focus group is considered a type of interview consisting of several respondents participating in the discussion of a topic [25]. The aim of a focus group is to facilitate a discussion and use that to interpret and understand the topic from the participants' perspective. It allows for interactions between the participants, which according to Johannesson and Perjons [25], enables a greater depth into the topic addressed compared to one-to-one interviews.

Our focus group was conducted in the first iteration as part of our evaluation activity. The participants consisted of us, our supervisors, and two members of the immuneML team. Beginning the focus group, we presented our work in the design and development activity. Following this, we used presented ideas and showed models to facilitate an open discussion regarding essential topics. The overall goal of the focus group was to reflect on what we had done thus far, create a discussion and generate new ideas to set the objectives for our second iteration.

3.9.3 Quasi-Experiment

A quasi-experiments is an empirical enquiry [55]. While experiments are based on randomization, the assignment of treatments to the subjects can not be based on randomization in a quasi-experiment [55]. Instead, the assignments emerge from the characteristics of objects or subjects. Quasi-experiments are, in other words, experiments where participants are not randomly assigned to experimental groups [28].

Goal

The quasi-experiment was the first evaluation method used in our second iteration of the final artifact and was the most extensive data collection for this project. The main goal of the quasi-experiment was to determine the effectiveness of the artifact in solving the proposed problem. This goal was directly related to the research questions, specifically RQ1 and RQ1.1, which focused on improving and simplifying extending a platform through an interface.

To evaluate the effectiveness of the artifact, we designed a quasi-experiment in which participants were divided into two groups, one using the old solution (Task A) and one using the new solution with the artifact (Task B). Both groups were given a task with the goal of extending immuneML and were asked to provide feedback on their experience. The old solution was used as a baseline for comparison with the new solution.

The objective of the quasi-experiment was to evaluate the artifact's usability by examining participants' user experience with the process of extending immuneML. Specifically, we assessed if they found the process understandable, easy to use, and satisfactory. By comparing the feedback of the two groups, we could assess whether the new solution improved and simplified the process of extending immuneML.

Participants

Background	**Number of participants**
PhD students	3
MSc and BSc students	4
Developers	3

Table 3.1: Table of participants in the quasi-experiment

Ten participants with different backgrounds took part in the quasi-experiment, as shown in Table 3.1. First, we collected data from PhD students from a craftsmanship session hosted at the Biomedical Institute at the University of Oslo. Secondly, we invited bachelor's and master's students to participate in our quasi-experiment. At last, we contacted developers with several years in the industry to participate.

The participants were split into two groups to perform Task A and Task B. We used assumptions about the participants' knowledge level to decide which task they should do as part of the experiment. The reasoning was to reduce the risk of having a low number of participants with a high imbalance in skill level. In the craftmanship session, this assumption was made by the meeting host, who already had knowledge of the participant's skill level. For the students, we assumed their skill level based on their progression in their studies, and for the developers, their years of experience.

Content of quasi-experiment

To gather data and feedback from participants, we designed a survey that presented the task and asked for their feedback. The details of this survey will be described in Section 3.9.5. The content of the quasi-experiment consisted of:

1. a questionnaire to identify the knowledge skills of the participants
2. a task description for the participants to perform
3. a questionnaire where the participants were asked how they interpreted the process of extending immuneML after performing the task.

Two different tasks were created for the quasi-experiment:

- **Task A**: using the old solution of extending immuneML
- **Task B**: using the new solution of extending the immuneML, containing our developed artifact

In both tasks, we provided participants with a step-by-step guide on how to extend immuneML by implementing a Logistic Regression, a classification model commonly used for linear and binary classification problems [43]. The survey for Task A is included in Appendix C, and the survey for Task B in Appendix D.

	Question	**ID**
Task description	I understood the purpose of the task	Q1
	The task description was clear	Q2
Development process	It was easy to follow the step-by-step guide for the task	Q3
	I understood how to add the new machine-learning method	Q4
	It was easy to add the new machine-learning method	Q5
	I understood how to run the analysis	Q6
	It was easy to run the analysis	Q7
	The process of extending immuneML was uncomplicated	Q8
	I am satisfied with the process of extending immuneML	Q9
YAML file	I understood what the YAML file was used for	Q10
	I understood the structure of the YAML file	Q11
	I understood where to fill in the necessary fields in the YAML file	Q12
	The YAML file was uncomplicated	Q13

Table 3.2: Post-task questionnaire for quasi-experiment

The post-task questionnaire, shown in Table 3.2, aimed to capture the participants' user experience of performing the task. Since our artifact changes how third-party developers extend immuneML with new functionality and run the platform, we wanted to see how the artifact affects the user experience of following the development process and using the YAML file. Tullis and Albert [46] presents the topic of self-evaluation in post-task questionnaires. They give examples of how one can formulate questions. We used this as inspiration to create questions that could be mapped to understandability, ease of use, and satisfaction.

The lists below show the mapping of the questions to what we wanted to measure:

Development process:

- Understandability: Q4, Q6
- Ease of use: Q3, Q5, Q7
- Satisfaction: Q9

YAML file:

- Understandability: Q10, Q11, Q12
- Ease of use: Q13

3.9.4 Survey with tasks

The research question RQ1.2 investigates ways to enable the integration of tools written in other programming languages with immuneML. To demonstrate this capability, we used the software Absolut (defined in Section 2.6.3) as our primary use case and illustrated how it could be used with immuneML through our artifact.

Goal

As a proof-of-concept, we did not have a baseline to compare with. Comparing an old and new solution was not possible due to the absence of this feature in the current immuneML version. To address this, we created a task similar to the quasi-experiment and conducted a corresponding survey, as described in Section 3.9.5. Instead of comparing two solutions, we used this data collection to assess users' experience in running an external tool with immuneML by looking at if they found the process understandable, easy to use, and satisfactory. The aim of this was to show the effectiveness of the artifact from a user's perspective.

Participants

Background	Number of participants
MSc and BSc students	2
Developers	2

Table 3.3: Table of participants in the survey with tasks

As depicted in Table 3.3, there are a total of four people participating in the data collection. These consisted of one master's and one bachelor's student and two developers from different companies.

Content

The structure of the data collection was the same as the quasi-experiment. The differences between the content of the quasi-experiment and this data collection were a different task description and a different post-task questionnaire, as shown in Section 3.9.5 and Appendix E. The content was as follows:

- a questionnaire to identify the knowledge skills of the participants
- a task description for the participants to perform
- a questionnaire where the participants were asked how they interpreted the process of using an external tool with immuneML

	Question	ID
Task description	I understood the purpose of the task	Q1
	The task description was clear	Q2
Process	It was easy to follow the step-by-step guide for the task	Q3
	It was easy to run immuneML with an external tool	Q4
	The process of running immuneML with an external tool was uncomplicated	Q5
	I am satisfied with the process of running immuneML with an external tool	Q6
YAML file	I understood what the YAML file was used for	Q7
	I understood the structure of the YAML file	Q8
	I understood where to fill in the necessary fields in the YAML file	Q9
	The YAML file was uncomplicated	Q10

Table 3.4: Post-task questionnaire for survey with tasks

The post-task questionnaire, shown in Table 3.4, aimed to capture the participants' user experience of performing the task. The questionnaire was created following the same approach as in the quasi-experiment, as described in Section 3.9.3.

The lists below show how we mapped the questions to what we wanted to measure:

Process:

- Understandability: Q5
- Ease of use: Q3, Q4, Q5
- Satisfaction: Q6

YAML file:

- Understandability: Q5, Q6, Q7
- Ease of use: Q8

Task description

The overall task description consisted of asking the participants to simulate the steps that a user would have to make to be able to run an external tool with immuneML, in this case, Absolut. The steps were as follows:

1. Download a new version of Absolut. This was an updated version of Absolut containing code inserted by us that made it possible to run from immuneML
2. Locate a file inside Absolut and make it runnable by running a makefile that is responsible for compiling the C++ code in Absolut
3. Download a partially filled-out YAML file
4. Fill in the YAML file following our documentation
5. Run immuneML with the YAML file

The full task description is shown in Appendix E.

3.9.5 Survey

A survey is a data collection method that gathers information from individuals by asking them to respond to a series of questions [36]. We used the same structure in our quasi-experiment and survey with tasks to present tasks and gather data. The structure was as follows:

1. An information page about the survey
2. A pre-task questionnaire
3. A task description
4. A post-task questionnaire

Information page

The initial section of the survey consisted of an information page where the motivation and purpose of the survey were presented.

Questionnaire

There are several ways of collecting data in a survey. One of the most common ones is questionnaires [36]. A motivation for using questionnaires is because of their efficiency. As Johannesson and Perjons [25] notes, questionnaires can efficiently gather opinions and perceptions about an artifact. An important downside, however, is that questions gathered can often be superficial [25]. This potentially blocks us from getting a deeper insight into the respondents' views. Still, since the task was already time-consuming, it would be more difficult to find participants willing to both solve a task as well as participating in an interview. As a trade-off, we included open-ended questions at the end of the questionnaire to get qualitative data and insight into their experience not being captured by the questions with the Likert scale.

In both the quasi-experiment and the survey with tasks, we used two questionnaires: a pre-task questionnaire to be filled out before the task and a post-task questionnaire to be filled out after completing the task.

Pre-task

The first questionnaire aimed to determine the participants' skill levels by asking them to assess their general programming skills, their proficiency in Python, their experience with machine learning, and their familiarity with immuneML. We included these questions because programming is a vital aspect of working with immuneML, Python is the programming language used in the platform, machine learning is essential to the context of the platform, and experience with immuneML helps us determine if participants have knowledge about the current process of extending the platform. Overall, these questions were included to ensure that the groups were comparable regarding experience when examining the data in our analysis.

Post-task

In the last part of the survey, participants completed a post-task questionnaire to gather feedback on their experience using immuneML and executing the task. The content of the post-task questionnaires is described in Section 3.9.3 for the quasi-experiment, and in Section 3.9.4 for the survey with tasks.

Likert scale

Our questionnaires consisted largely of statements used for self-estimation. To capture the self-reported data, we used rating scales, which according to Tullis and Albert [46] is one of the most common ways to capture self-reported data when studying user experience.

We used Likert Scales, which is one of several approaches that can be used. We used a five-point scale with the options strongly disagree, disagree, neither agree nor disagree, agree, and strongly agree. Several approaches to this exist. We chose to use Likert Scales. According to Tullis and Albert [46], using such a traditional Likert Scale with the options mentioned above should provide a measure of usability in the context of asking participants to rate the ease/difficulty of tasks.

Quality check of surveys

To quality check our survey, we did a pilot test. As Roopa and Rani [40] mentions, pretesting of a questionnaire is done for several reasons: to determine whether the questions are framed properly, whether the wording of the questions will achieve the desired results we want if they have been placed in the right order, that the questions are easily understood if there is a lack of questions or some that should be removed, and at last to determine if the instructions are satisfactory.

3.9.6 Semi-structured interview

An interview is a communication session between a respondent and a researcher, where the latter has control of the agenda and asks questions to the responder [25]. A semi-structured interview is a type of interview where questions can be discussed in a more flexible order. Unlike structured interviews, the questions can be open, allowing the respondents to formulate themselves using their own words.

We decided on a semi-structured interview because we wanted specific data on our fully implemented artifact. We, therefore, saw it fitting to conduct a semi-structured interview allowing both an open discussion and, at the same time, ensuring that specific data were collected. Alternatives could have been performing a structured- or unstructured interview. However, we saw that a structured interview would limit us too much to specific questions and risk losing feedback that could be gained from a discussion around the questions. On the other hand, an unstructured interview could prevent data from being caught because the conversation would be too open.

Before the interview, we created an interview guide, which is a list of questions used during an interview to direct the conversation toward the research topic [27]. The guide was used to establish an overall structure for the interview, ensuring that all essential topics were covered to collect the required data. The interview guide can be found in Appendix F.

We also prepared a consent form, see Appendix B, where we introduced our research topic and why we conducted the interview. As a formality, the consent form informed the participants that we would be taking notes during the interview.

3.10 Data analysis

The data collected for this study consisted of both quantitative and qualitative data. Our quantitative data was collected through a quasi-experiment and a survey with tasks. Most of our qualitative data was collected through the focus group and the semi-structured interview. However, we also gathered some qualitative data from the quasi-experiment and the survey with tasks through the post-task questionnaires, as presented in section 3.9.

To analyze the qualitative data from the focus group and interview, we conducted a thematic analysis. The quantitative data were analyzed through descriptive analysis.

3.10.1 Thematic analysis

Thematic analysis is a method for identifying, analyzing, and interpreting patterns (themes) of meaning within qualitative data [12]. Minimally, thematic analysis organizes and describes data in detail [8]. As Braun and Clarke [7] describes, thematic analysis is used because of its accessibility and flexibility. Our selection of this method is based on their description of thematic analysis as a good entryway into qualitative research.

The thematic analysis separates between two different approaches: an inductive and deductive approach. By following an inductive approach, the data that is coded and analyzed is done through a bottom-up approach driven by the content of the data [7]. This initially means the codes and themes are derived from the data content. On the other hand, a deductive approach follows a top-down approach for the data coding and analysis [7]. This means that a series of concepts, ideas, or topics are brought to the data used for coding and interpreting the data. In other words, codes and themes derive more from what the researcher brings to the data [7].

Our thematic analysis followed a deductive approach as our codes and themes were derived from our research questions and ideas. To conduct the analysis, we followed an approach with six phases outlined by Braun and Clarke [8].

1. We familiarized ourselves with our data by going through the notes taken during our focus group and interview, reading and re-reading the data
2. We generated codes using our research question and topics explored
3. We searched for themes based on our initial codes related to our research questions as well as patterns in the data
4. We reviewed the themes, ensuring that they were relevant and coherent with our data
5. We defined and named the themes in a way that reflected the meaning and relevance of our research question

6. We reviewed and refined the themes

3.10.2 Descriptive analysis

To analyze our quantitative data, we conducted a descriptive analysis. Descriptive analysis is a statistical method used for summarizing raw data from a sample or population [29]. According to Fisher and Marshall [13], descriptive analysis is one of the easiest types of statistical analysis to both perform and interpret.

Our analysis involved using visualized data to summarize and describe the key features of our data as the basis for interpreting the data. We chose to use diverging stacked bar charts and box plots as our visualization techniques, which according to Saffo, South and Worth [41], performs well in presenting Likert scale data accurately. The box plot allowed us to visualize the experience level of participants. The diverging stacked bar chart allowed us to compare the distribution of positive and negative responses across statements related to understandability, ease of use, and satisfaction.

Chapter 4

Iteration 1

This chapter begins by introducing the first step in the DSR process, which is identifying the problem and motivation. We then provide a comprehensive overview of the first iteration in our DSR process, outlining the work completed for each activity and the findings from the evaluation.

4.1 Identify problem and motivation

To begin the problem identification and motivation activity, an informal meeting was organized with members of the immuneML team early on in the thesis project. The purpose of the meeting was to discuss potential areas in immuneML that could have potential improvements and be explored from a software engineering perspective.

The immuneML platform, currently being developed solely by the immuneML team, aims to enable third-party developers to create tools and extend the platform. As a platform, it is essential for immuneML to be flexible and extensible, allowing for the development of new features and functionalities that can build on top of the core. However, extending the platform comes with several challenges, both for the team and external developers. The immuneML team expressed an interest in improving the process and creating best practices to develop the immuneML project further. This outlined the overall research question, RQ1, as well as RQ1.1.

Another point raised was that existing and new tools could be written in programming languages other than Python. E.g., a tool could be written in C for better performance. It would be very beneficial if the platform supported using tools in any programming language. It would remove the obstacle of only using tools written in Python, making it easier for developers to reuse existing projects. This was used to derive the research question RQ1.2.

Following this discussion, it was concluded that improving the extensibility of immuneML and allowing the use of external tools was the most appropriate problem to explore further, additionally allowing for them to be written in different programming languages.

Exploring the topic

To further explore the problem area, we conducted a literature review with a considerable focus on extensibility. We also looked into literature about the versatility of programming languages used in software within the field of bioinformatics and technologies that can be used to enable extensibility. The results of the literature review are presented in Chapter 2.

Development process

To understand and identify areas for improvement in the development process, we modeled the process, as depicted below in Figure 4.1. The process for making contributions and extending the platform was extracted from immuneML's documentation [22]. This is a descriptive process model, a representation showing the actual process [32].

Through this analysis, we identified various problems and improvement areas:

- Extending the immuneML platform with new functionality requires a development process that involves understanding and editing the source code directly. Although the platform provides comprehensive documentation on how to extend it, this process can be challenging and time-consuming due to the platform's relatively large codebase. Moreover, each contribution to the platform increases the size of the source code, which potentially, over time, can make it more complex to maintain and manage.
- In order to make the new functionality available to other immuneML users, a third-party developer must submit a pull request for review by the immuneML team. The team then assesses and approves the contribution based on certain standards. If revisions are needed, the immuneML team provides feedback for necessary changes. This process may involve multiple iterations until the contribution meets the required standards and is approved. However, as the number of contributors to the platform increases, this process can become time-consuming for both the contributors and the immuneML team.

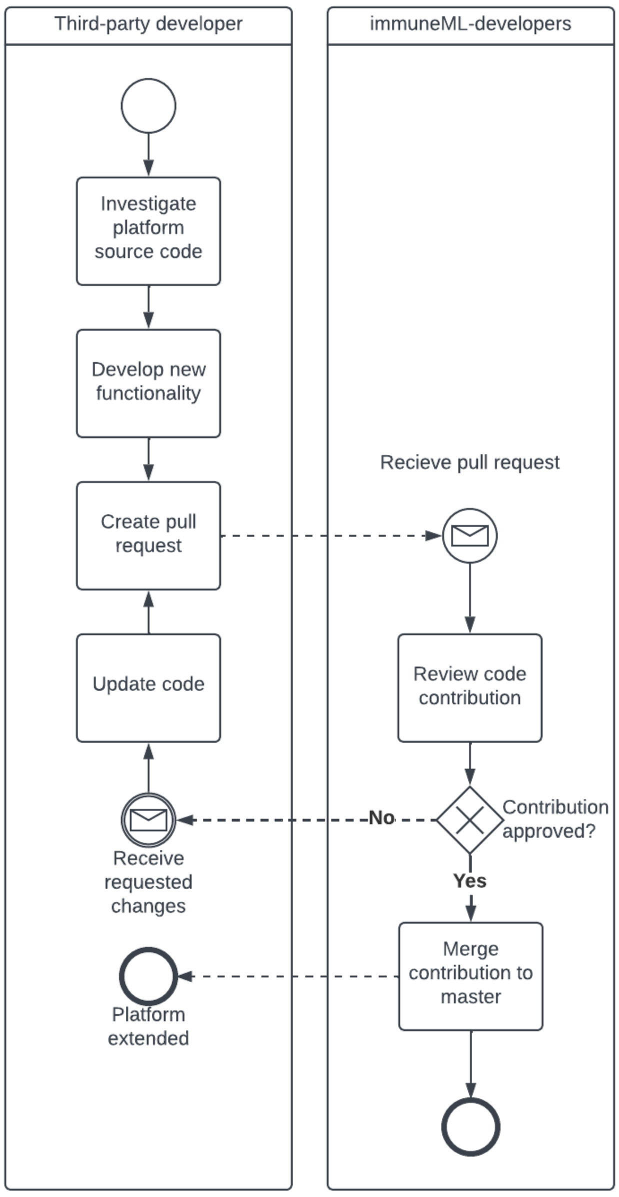


Figure 4.1: Descriptive BPMN process model for extending immuneML.

4.2 Objectives

In this section, the objectives for the first iteration will be outlined. These objectives were created based on our initial meeting with the immuneML team, our understanding of the problem, and our interest in exploring ideas and possible solutions.

Our initial goal was to look at extensibility and how this could be improved for the immuneML platform. Before looking at how we could improve it, we first had to build a proper foundation. We, therefore, set the objectives for this iteration to gain a practical understanding of immuneML and extend it on a small scale. The work we did was split into two parts:

Firstly, the goal was to extend immuneML by integrating a machine learning framework not currently implemented in the platform. This objective would be achieved by utilizing the current extensibility mechanism of immuneML, which involves using its abstract classes in a glass-box approach.

Secondly, enable the use of Python classes outside the immuneML package. The goal of this task was to discover how we could extend immuneML by introducing code located outside of the immuneML software core.

The objective of this iteration is to use the tasks to produce models and prototypes to build a foundation for further development. By evaluating them, the feedback is used to build the objectives for the next iteration.

4.3 Design and development

As the first iteration in our DSR process, the design and development activity was influenced by the fact that we had to gain an understanding of the platform. This required a thorough analysis of the platform source code. Through analyzing the documentation, running simple examples, and stepping through the process, we were able to get a good grasp of the key concepts.

Our design and development process can be separated into three steps, presented in Sections 4.3.1, 4.3.2, and 4.3.3.

4.3.1 Import classes outside the core

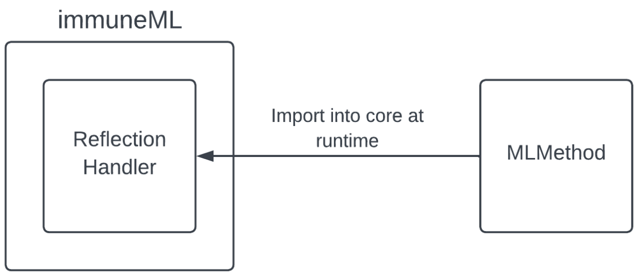


Figure 4.2: Importing ML methods outside into the core

The first step consisted of importing Python modules from outside the core. This was achieved by adding functionality to import classes located outside the core, using an already existing module in immuneML that handles internal classes, as shown in Figure 4.2. Using the same abstract methods and naming convention of files and folders already used by immuneML, minimal changes to the logic of the process and source code were needed. When running the program, the user would have to specify the location of the external files as an argument to the command line interface.

The step produced a functional prototype that had several limitations. Firstly, all external classes had to be in the same folder, and secondly, these classes had to inherit from internal abstract classes within immuneML. This meant developers would need to understand the platform's internal data structure to extend the platform. In summary, while this prototype allowed for external code, it did not offer any other advantages for improving platform extension.

4.3.2 Implement ML frameworks following the current process

The second step involved integrating third-party ML frameworks into immuneML using the existing platform extension process. We gained insights into how to extend the platform through the platform's documentation and abstract classes. The selected frameworks were PyTorch Tabular and TensorFlow, which had not been previously utilized in immuneML. PyTorch Tabular is based on the PyTorch deep learning library and provides a more accessible approach to using deep learning with tabular data [26]. TensorFlow is a complete machine learning platform [45], which was only used for training an ML model.

4.3.3 ToolParser and YAML file

In the last step, we redesigned how the user would define external code to use the YAML specification file instead of the command line interface. This was motivated by the need for reproducibility in the analysis process. Defining an analysis through the command line interface was unnecessarily complicated and did not facilitate the same level of reproducibility as using the YAML file.

Several solutions for specifying external code in the YAML specification file were investigated. One solution would be to define the external code where it is to be used in the definitions and instructions. However, we interpreted that this would add more complexity. Therefore, we decided to introduce an entirely new section in the YAML file specifically for specifying external code, named specifying "tools." Our idea behind this was that it could show a more clear separation between what already exists inside of immuneML and what is run outside of immuneML.

```yaml
tools:
  my_tool:
    path: /path/to/folder
    language: Python
    type: MLMethod
    name: my_method
```

(a) One type

```yaml
tools:
  my_tool:
    path: /path/to/folder
    language: Python
    types:
      - type: Preprocessor
        name: my_preprocessor
      - type: MLMethod
        name: my_method
```

(b) Two types

Figure 4.3: Example of YAML specification for a tool

Our proposed changes to the structure of the YAML file, including a tool section, are shown in 4.3. It shows two versions. Figure 4.3a displays defining a tool with one type, while Figure 4.3b shows how a tool can define multiple types. For each tool, there are four parameters:

- **path:** path to the folder where the tool is located
- **language:** the programming language that the tool is written in. It was not used but included to demonstrate how the user could specify the language for the tool in future versions.
- **type:** the type of tool is determined by this parameter, which is based on its intended purpose. Our use cases involved training ML models and preprocessing datasets, so the parameter was set accordingly. We provide examples in the figures that demonstrate how a tool can be defined for both training ML models and preprocessing.
- **name:** the name of the tool type, used as an identifier for the use of the tool in the definitions and instructions sections.

4.3.4 Design sketches

This section will describe the models and ideas that we created in the design and development activity. These models were used in the following evaluation activity to get feedback for further development.

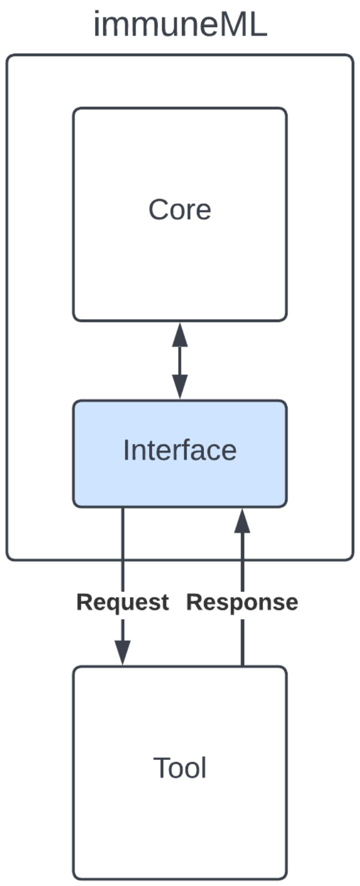


Figure 4.4: Design sketch of a solution using an interface

As described in Section 4.3 our prototype has several limitations, e.g. high coupling. We, therefore, saw the need to separate the core and the external code completely by introducing an interface. A sketch of this is shown in Figure 4.4.

The platform package will include the interface that facilitates communication between the tool and the platform core. The interface serves as a mediator that enables immuneML to send requests and receive responses by external tools through the interface. Because of the request and response structure, the question of whether or not these should be synchronous or asynchronous therefore arised.

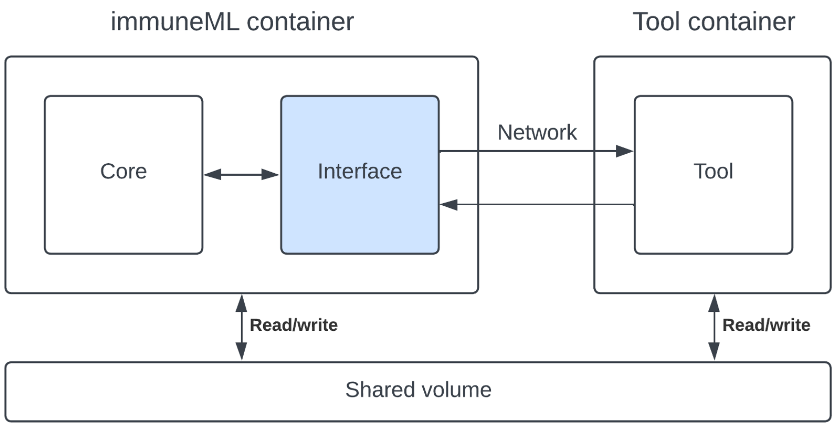


Figure 4.5: Design sketch of a solution using Docker

One idea we investigated was the use of container technology. We considered using container technology, specifically Docker because it provides a way to package applications and dependencies into containers. By containers in Docker, we refer to a unit commonly used for packaging code and its dependencies, enabling applications to run efficiently and dependably across different computing environments [52]. The user would not have to set up a virtual environment and install the required dependencies when running immuneML and tools written in Python.

Figure 4.5 shows a sketch of a possible solution using Docker. The immuneML platform and tools would run in separate containers and communicate using the virtual network in Docker. The shared volume in Docker could be used if large files were to be transferred, like a trained ML model. The data has to be readable and writable for both programming languages. This could be solved using a common data structure like JSON, CSV, or TSV files.

4.4 Evaluation results

This section will describe the demonstration and evaluation carried out during the first iteration through a focus group. These activities aimed to gather feedback and thoughts about our current work and ideas from members of the immuneML team. First, we present our demonstration, followed by the results from our thematic analysis of the data gathered from the focus group.

4.4.1 Demonstration

During the demonstration, we presented the tasks, consisting of incorporating code from outside the immuneML source code. We presented the prototypes and models described in Section 4.3. We also presented a proposed structure of the YAML file that we saw fit for our solution, as described in Section 4.3.3. We provided a walkthrough of the YAML structure, highlighting the various components and how they fit into the overall design. We explained our design decisions and how our proposed solution addressed the identified problems.

Throughout the demonstration, we encouraged the team to ask questions and provide feedback on our proposed solution. The feedback we received was incorporated into the ex-ante formative evaluation, allowing us to further refine our proposed solution. Overall, the demonstration allowed us to showcase our progress and gather valuable feedback from the immuneML team.

4.4.2 Prioritization of objectives

During the focus group session, we discussed the prioritization of objectives for the project. One important question was which stakeholder should be the primary focus; should it be the developers of immuneML, the users, or third-party developers? The immuneML team suggested that there should be less focus on the immuneML developers and more on users and third-party developers. One team member stated that the least important is the immuneML developers, and continued with *"... encouraging users and third-party developers to join is more important"*.

Besides the prioritization of users, the team emphasized that enabling the use of tools written in different programming languages would allow for the most dramatic change and therefore proposed to be something to focus on. They proposed that this was something that should be prioritized in our next iteration.

4.4.3 YAML file

The response to our proposed YAML structure was that thought it looked good from a logical perspective. However, they were unsure about how a user would perceive the structure. The feedback on a separate section for tools was *"that seems intuitive"*. One specific improvement they mentioned was the naming convention, which they found confusing.

4.4.4 Design sketches

We presented and discussed the design sketches, as depicted in section 4.3.4, and our ideas regarding solutions. We asked whether or not our solution should be asynchronous or synchronous and received clear feedback that we should stick to synchronous in our solution. They did not believe creating a solution supporting asynchronous calls would be necessary.

We also discussed the use of container technology, such as Docker. immuneML encourages using Docker and are providing a Docker image for the platform. While they looked at the proposed solution in a positive way, they did not believe this would be a good fit for this project and its users. Their concern was that setting up Docker would be too much for an average user, such as bioinformaticians.

4.4.5 Use cases

During our initial discussion with the immuneML team prior, we explored the use of Absolut as a potential use case for further work. We were interested in understanding how Absolut, a program written in a different programming language than immuneML, could be integrated into the immuneML platform. The team suggested that we start by allowing Absolut to be run by YAML, enabling the program to be used as a separate outermost-level component. They also discussed how Absolut's use in immuneML is not tightly integrated with the rest of the core immuneML, making it easier for users to install and use. The team mentioned that Absolut is a tool that could be used to prepare a dataset for later use in immuneML. They suggested that this use case could be a step towards making the installation and use of different programs with immuneML easier for users in the future. Based on this, we identified that our further work with Absolut should be on preprocessing datasets.

Similar to our discussion regarding Absolut, we also discussed using DeepRC with immuneML. While our discussion was less focused on this during the focus group, our interpretation was that DeepRC is something of interest to the immuneML team and, therefore, a use case appropriate for showing how immuneML could run training of machine learning models externally.

Chapter 5

Iteration 2

This chapter provides a detailed description of our second iteration, outlining each activity. As the final iteration, this chapter presents the final evaluation of our artifact.

5.1 Objectives

The overall objective for this iteration is to develop a fully working artifact that explores our research questions. First, we present the feedback from our first evaluation, presented in Section 4.4. We use this to specify what should be improved and focused on when designing and developing the final artifact in the second iteration. Last, we present an overview of what the artifact should accomplish.

Focus and improvements based on first evaluation

- Prioritize users and third-party developers
- Allow the use of tools written in different programming languages
- Improving YAML structure
- Simplify naming conventions

Defining what the artifact should accomplish

Based on the first evaluation, we identified parts of our solution that should be improved in the second iteration.

In addition to identifying focus improvement areas for this iteration, we set requirements for what the artifact should accomplish to answer our research questions. These are listed in Table 5.1. For each requirement, we identified key quality attributes essential for achieving that requirement and to help guide our decisions during the design and development process. These quality attributes are extensibility, usability, interoperability, and maintainability. The definition of these attributes is presented in Chapter 2 under Section 2.2.1.

What would the artifact accomplish?	**Quality attributes**
Encapsulation of complexity of tool integration	Extensibility, usability
A design pattern with relatively low complexity	Extensibility, usability
Enable the use of tools written in other programming languages	Interoperability
Internal changes to the core will not directly affect the tools	Maintainability

Table 5.1: Requirements for the artifact and connected quality attributes

5.2 Design and development

To improve how immuneML is extended, our proposed solution encapsulates the immuneML core, inserting black-box extensibility as a new extensibility mechanism. In this section, we present models depicting the new process for extending immuneML, our design decisions, and use cases used to define the interface's functionality and demonstrate how it serves its purpose.

5.2.1 Process changes

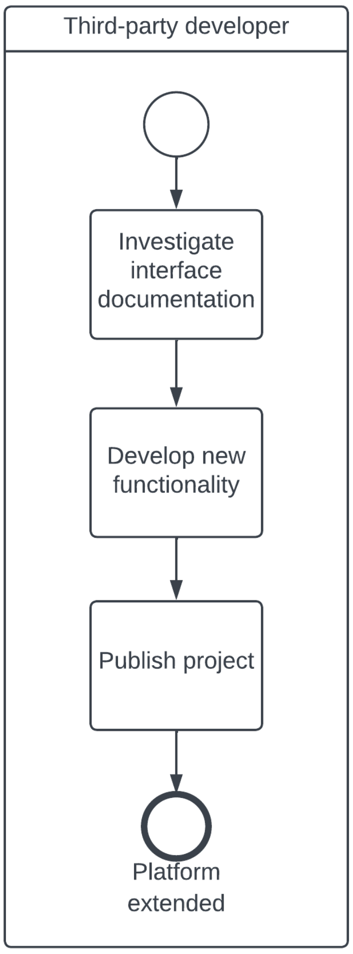

Third-party developer
Investigate interface documentation
Develop new functionality
Publish project
Platform extended

Figure 5.1: Prescriptive BPMN process model for extending immuneML

The goal is to simplify the development process by encapsulating the core with an interface for developers to integrate their tools, eliminating the need to modify the ImmuneML source code and allowing developers to focus on mastering the interface. The introduction of this interface would streamline the development process.

A prescriptive process model of the changes introduced for extending the platform for a third-party developer is illustrated in Figure 5.1. In contrast to the descriptive process model, depicted in Figure 4.1, the developer no longer depends on the immuneML team to make their contribution available to other users. The developer must make their tool available for users, e.g., publish it on GitHub. Additionally, the developers of immuneML would no longer have to be involved in reviewing and approving new contributions.

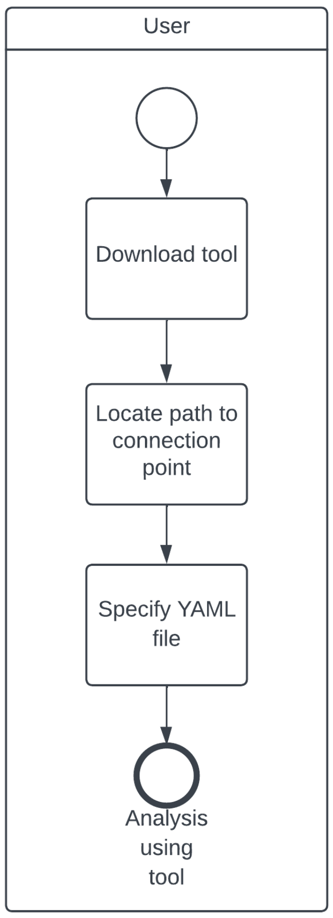


Figure 5.2: Prescriptive BPMN process model for using immuneML with tools

The integration of external tools on the platform introduces additional steps for users. The new process, depicted in Figure 5.2, involves downloading the tool from a source such as GitHub or a webpage, locating the tool's path and connection point, and then defining the analysis using the YAML file as usual.

5.2.2 Design decisions

During the phase of designing the solution, our focus was on the quality attributes described in Table 5.1. For each design decision, we investigated possible solutions and narrowed them down to what we believed was the best solution based on the requirements. In this section, the process of the design decisions will be described. The final artifact will be described in Chapter 6.

The topics we present regarding our design decisions are as follows:

- **Calling functions directly from the immuneML core:** we explored ways of calling functions outside of immuneML directly

- **Running tools as a subprocess:** we explored how tools could be run as a subprocess of immuneML
- **Establish communication with the tool:** we looked at how communication with tools could look like
- **Data sharing:** we explore different mechanisms of how data between immuneML and tools could be shared
- **Use of task schedulers:** we explored the use of having a task scheduler as part of our solution
- **Design of the architecture:** we looked at different ways of designing an architecture for the interface
- **Structure and parsing of the YAML file:** we looked at what type of structure the YAML file should have

Call functions in tools directly from immuneML

Several libraries and packages facilitate the functionality to call functions in other programming languages directly from Python [23]. This approach could simplify the process for third-party developers since they only need to create the functions defined by the interface with minimal modifications. However, the drawback of this approach is that it requires one implementation for each programming language. Multiple implementations would be required to support numerous programming languages, resulting in an extensive interface that would be difficult to maintain. We concluded not to explore this option further.

A second approach is to use RPC, which can call functions in another language over a network interface [6]. Through a web service API, the function call gets translated into a language-independent format and executed on the server, with the results being returned to the calling program. [6]

One of the downsides of using RPC as a solution is that it requires third-party developers to understand and implement RPC in their projects. This would not only contribute to reducing usability and integrability, but it would also not support the aim of our RQ1.1, which is about how extending immuneML can be simplified. Given that immuneML and its tools will be running locally on a computer, we concluded that the benefits of using RPC are not significant enough to justify its implementation. Therefore, we did not go further with this option.

Subprocess

To run tools on a local computer, we explored options for running them as an external process from immuneML directly. We found that the subprocess Python module provides a solution for launching and managing processes while connecting to their pipes [44]. The module can run executables or scripts by providing their path. The subprocess module uses the underlying functionalities of the operating system to spawn processes and automatically defaults to different system calls depending on the operating system in use. In summary, we can use the module to initialize tools in any programming language as an external process, communicate through pipes, and it is supported on different operating systems.

We tested different solutions by creating simple prototypes of a project running external code as child processes. By investigating how the solution could be used with our use cases, we saw that pipes had some limitations. In the use case of DeepRC, an ML method, we want to continue to use the same process in immuneML for training models and performing hyperparameter optimization. The tool must be called multiple times and run different functions to achieve this. This meant we needed a way to communicate with our tool multiple times and at different steps of the immuneML analysis process while running the process in the background. The communication between the parent process and subprocess in the Python subprocess module occurs only during process start and termination, so we couldn't solely depend on the module.

We saw that using the subprocess Python module was beneficial and decided to use this as a foundation. However, it needed to be supplemented with a solution for better communication.

Communication

Based on the limitation of using the Python subprocess module for communication, we had to look into ways of establishing communication between the processes. One way to communicate between processes is by messaging libraries. This section presents our process of determining an appropriate communication solution.

To implement the interface to connect tools to the platform, we compared several messaging solutions to identify the most suitable option. Our evaluation was based on two primary factors: interoperability and usability. These quality attributes are defined in Section 2.2.1.

Interoperability was a key consideration for us, as we needed the messaging library to work across multiple programming languages and operating systems. We evaluated each library based on its ability to operate with different message formats and protocols and its ease of integration with other tools and systems.

Usability was also critical in our evaluation, as we wanted the messaging library to be easy to use and intuitive. We assessed each library based on its ease of setup and configuration and the quality of documentation.

To find an appropriate messaging library, we explored several resources, such as documentation of libraries and developer communities like Stack

Overflow. Through this process, we created a table summarizing and highlighting the strengths and weaknesses of the findings, which can be seen in Appendix G.

Based on our exploration, we decided to use ZeroMQ, defined in Section 2.6.2. The library's high level of interoperability allowed us to communicate seamlessly across programming languages and work on different programming languages. We also found it easy to use, with minimal implementation required to set up communication in various programming languages. The library's lightweight, low-latency messaging architecture was another major factor in our decision, as it allowed us to achieve fast, efficient communication between our components. One of the drawbacks, however, is that it does not provide error handling when for instance, establishing connections. This is one of the trade-offs resulting from its focus on simplicity. Overall, we determined that ZeroMQ was the best fit for our project due to its strong performance, ease of use, and broad support for multiple programming languages.

Data sharing

Initially, we used ZeroMQ and Pickle data, a serialized Python object, to share data with the tool. However, while this approach proved helpful for prototyping and developing solutions, it had limitations. Tool developers would need to thoroughly understand immuneML's internal data structure, and the data was not interoperable with other programming languages. We tried to address this issue by serializing the dataset into JSON format. However, this approach still required the tool to deserialize the data and depended on immuneML's internal structure. Sending large datasets and trained models over sockets could also result in poor performance, making it problematic to use this approach.

We decided to use file sharing and immuneML's built-in support for exporting and saving datasets to files, explicitly using the standard data format TSV. This solution made the data interoperable across several programming languages, removing the dependency on immuneML's internal structure. Another advantage of TSV files is that they are human-readable, making it easier for third-party developers to understand the data they are working with.

The final solution for our data-sharing involved using sockets with ZeroMQ to communicate with the tool. This allowed for efficient and flexible communication between processes. The communication was carried out using JSON format, which included instructions for the tool and the path to the dataset in TSV format. Once the tool finished its task, it could either return the path to the resulting data or save it directly into the result folder for the analysis run. This solution provided us with the necessary control and flexibility to execute our tools effectively.

Task schedulers

We explored options for running tools as subprocesses using task schedulers to provide efficient resource allocation and enable us to manage and monitor the execution of running tools. We looked into the task schedulers Prefect, Celery, and Airflow. However, we discovered that this was not something we wanted to use as part of our solution. The reason why was that for the current design of the interface, using the subprocess module in Python directly already provided us we the flexibility and control needed to use our tools effectively. We also saw that introducing a task scheduler could introduce more complexity for this purpose.

Architecture

When designing the architecture, our primary focus was on maintainability. We separated the main parts of the changes into a new and separate package and created a single point of communication between the core and this package through a component called controller. This controller acts as the interface to the package and handles all requests.

We also made it easy for developers to expose new functionality for the interface. With minor changes to the core, developers can add an abstract class and then extend the interface with a new component. This makes it possible to add new features without having to modify the existing codebase extensively, which should help to ensure that the platform remains maintainable over time. The architecture will be described in detail in Chapter 6.

ToolParser and YAML file

```
tools:
  my_method:
    path: /path/to/folder
    type: MLMethodTool
```

(a) One type

```
tools:
  my_preprocessor:
    path: /path/to/folder
    type: PreprocessorTool
  my_method:
    path: /path/to/folder
    type: MLMethodTool
```

(b) Two types

Figure 5.3: Example of YAML specification for a tool

As a result of the feedback from the immuneML team, we made several changes to the YAML file and ToolParser. Firstly, we streamlined the structure of the tool section by using only one name for each tool instead of two. However, this change has the downside of requiring users to specify the path multiple times if a tool is used for more than one task. Secondly, we reduced the indentation when specifying multiple types for a tool. This maintains the same structure for tool sections and enhances readability. Thirdly, we eliminated the need to specify the tool's programming language, as the ToolParser will automatically handle this information. Finally, we added validators to the ToolParser to ensure that all keys in the YAML file were valid, all required keys were present, and the tool type was valid. Figure 5.3 shows an example of the new solution.

5.2.3 Use Case 1: Absolut

Our thesis presents a use case in which Absolut, a bioinformatics tool written in C++, was used to demonstrate the ability to run external tools written in other programming languages with immuneML through our developed interface. The use case aimed to showcase how immuneML can utilize external tools to preprocess a dataset generated by immuneML, which Absolut then extended by utilizing the information from the dataset before importing it back into immuneML. We further collected data on how users interpreted using Absolut with immuneML. The results from this evaluation are presented in Section 5.3.2. The link to the source code for this use case can be found in Appendix A. Our development process in this use case followed four steps:

Step 1 - Run a program in a different language

The first step was to make immuneML run a program written in a different programming language through the command line. This was done using the Python subprocess module. To test this functionality, simple and small programs were created to simulate preprocessing a dataset from immuneML.

We first created a Python script to demonstrate how immuneML can interface with external programs. The script read the exported data in tabular format, generated prime numbers up to the number of rows in the table (subtracting the header), and then added the prime numbers as a column, appending it to the dataset from immuneML. Finally, the script returned the path to immuneML by printing it out in the terminal.

Step 2 - Implement programs in Java and C++

Following successfully implementing the Python program, the same program was created in both Java and C++. This was done to see if the interface could run programs in different programming languages. This turned out to work because the Python subprocess module is able to run any file as long as the file can be executed on the operating system.

Step 3 - Introducing ZeroMQ

In the third step of development, we encountered an issue with communicating with subprocesses while running. The subprocess module in Python only allows communication through pipes during process start and termination. However, we saw it more fit to have a solution where communication is not so limited but rather available throughout the entire process. We also saw that passing messages through pipes was relatively error-prone. As a solution, we introduced ZeroMQ to enable a better way of communicating between processes. Utilizing ZeroMQ, we created a script for each program following the same structure for how to connect to immuneML, resulting in blueprints for how to connect to immuneML, shown in appendix I.

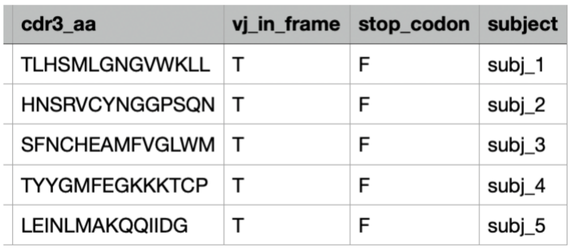
Step 4 - Implementing for Absolut

The results from development step 3 were adapted to work with Absolut. This step was considerably larger than the previous steps as a result of us having to go through the Absolut source code and understand its structure. We also had to learn the basics of C++ to be able to work with the Absolut source code.

The result of this step was that we were able to use the same C++ blueprint used for implementing the small programs for adding primes to the dataset. By using the blueprint, we extracted the necessary functions from Absolut and added them into the section instructed by the blueprint, resulting in a connection script. This was then turned into an executable to make it runnable. This step resulted in a working implementation.

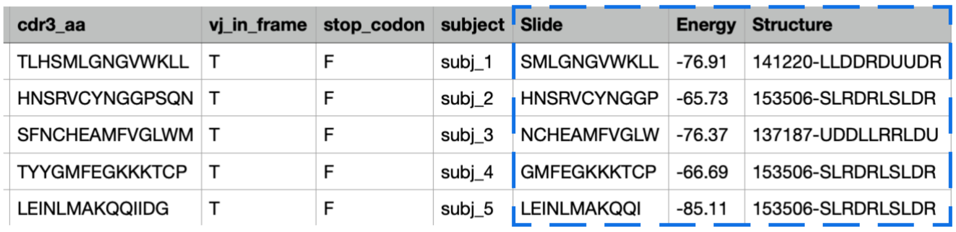
Resulting data

The data from immuneML and the results are illustrated in two figures. 5.4 shows the last four columns of a randomly generated immune receptor dataset generated by immuneML. The content of this dataset is outside of our understanding as we are not domain experts. However, the most essential part here is that the content under the column "cdr3__aa" was sent into Absolut, which in return created data that were converted into a table in TSV format. This table was further merged together with the data from immuneML, resulting in the table displayed in Figure 5.5.



cdr3_aa	vj_in_frame	stop_codon	subject
TLHSMLGNGVWKLL	T	F	subj_1
HNSRVCYNGGPSQN	T	F	subj_2
SFNCHEAMFVGLWM	T	F	subj_3
TYYGMFEGKKKTCP	T	F	subj_4
LEINLMAKQQIIDG	T	F	subj_5

Figure 5.4: Dataset from immuneML in TSV format before preprocessing with Absolut



cdr3_aa	vj_in_frame	stop_codon	subject	Slide	Energy	Structure
TLHSMLGNGVWKLL	T	F	subj_1	SMLGNGVWKLL	-76.91	141220-LLDDRDUUDR
HNSRVCYNGGPSQN	T	F	subj_2	HNSRVCYNGGP	-65.73	153506-SLRDRLSLDR
SFNCHEAMFVGLWM	T	F	subj_3	NCHEAMFVGLW	-76.37	137187-UDDLLRRLDU
TYYGMFEGKKKTCP	T	F	subj_4	GMFEGKKKTCP	-66.69	153506-SLRDRLSLDR
LEINLMAKQQIIDG	T	F	subj_5	LEINLMAKQQI	-85.11	153506-SLRDRLSLDR

Figure 5.5: Dataset from immuneML in tabular format after preprocessing using Absolut

Limitations

This use case served as a proof-of-concept, and it is important to acknowledge its limitations in terms of usability. The focus of this use case was to simulate the preprocessing of a dataset, where a dataset generated by immuneML was exported in tabular format and then modified by adding new data to it using Absolut. The resulting dataset was then imported back into immuneML. However, it is important to note that the added data cannot be used further in immuneML as there is no functionality in immuneML using such data. Therefore, this use case was strictly aimed at demonstrating the potential of our interface to enable the use of tools written in different programming languages.

5.2.4 Use Case 2: DeepRC

The second use case was to extend immuneML with an ML method tool, DeepRC. As described in Section 2.6.3, DeepRC is an ML method based on deep learning architecture for immune repertoire classification. The method was already implemented into immuneML as a package, using the old solution of extending the platform. However, the integration uses an older version of the package since the immuneML team has not had the resources to update to the newest version of DeepRC. This is an example of how the current mechanism for extending immuneML negatively affects integrated code.

To connect DeepRC to immuneML, we had to create an interface for the project. We used the project's source code to construct an interface that relied on DeepRC's internal functionality, allowing us to connect it to immuneML. This interface enabled us to run functions from the immuneML model training process, such as fit and predict, in DeepRC to train a model. By using this approach, we were able to demonstrate the effectiveness of our solution for extending the platform with tools and confirmed that the final artifact was operational. The link to the source code for the implementation of this use case can be found in Appendix A.

5.3 Evaluation results

In this section, we will discuss the results obtained from the evaluation of this iteration. To collect the data, we performed a quasi-experiment, a survey with tasks, and a semi-structured interview. We will present the data from these collections separately, along with our interpretations of the results. Each section will contain a summary of the findings.

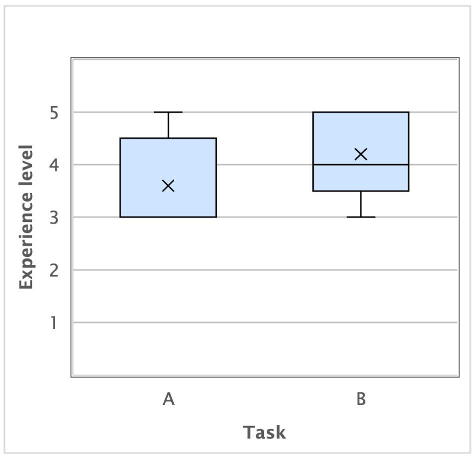
5.3.1 Data collection 1: Quasi-experiment

In the quasi-experiment, we conducted a comparison between the current solution for extending immuneML and the new solution, which involved using the artifact to extend immuneML. To evaluate how the new solution was perceived, we divided the participants into two groups based on the task they were assigned to perform.

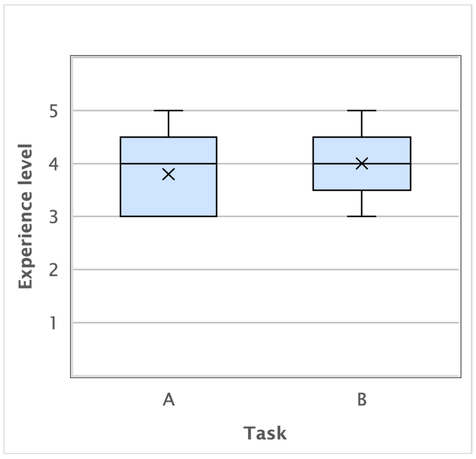
- **Task A:** using the old solution of extending immuneML
- **Task B:** using the new solution of extending the immuneML, containing our developed artifact

The content of the quasi-experiment and description of how we conducted it is described in Chapter 3 under Section 3.9.3.

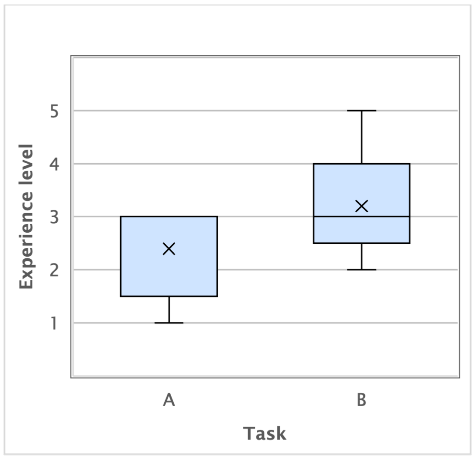
Experience of participants



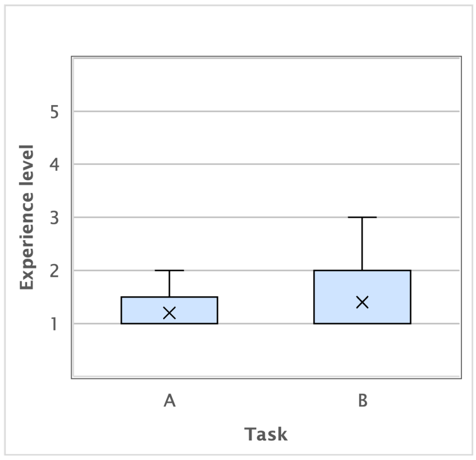
(a) Programming



(b) Python



(c) Machine learning



(d) immuneML

Figure 5.6: Participants' self-assessment of experience. Scale: No experience (1), beginner (2), some experience (3), intermediate (4), advanced (5)

First, we will provide a descriptive analysis of the box plots generated from the Likert scale survey questions. The questions were related to the participants' experience with programming, Python, machine learning, and immuneML. The charts provide an overview of the participants' background and experience levels. The data indicates that the experience levels of the participants are generally even across all groups. However, there is a recurring trend for participants performing task B to have slightly higher levels of experience.

We believe that the experience level with the most impact on task execution is programming experience since they are performing a development process. Machine learning and immuneML experience are less important for the task but are relevant for understanding the context of what they perform by doing this task. The distribution of experience levels is relatively even across all groups, except for machine learning, where the participants performing task B have a higher level of experience.

Understanding of tasks

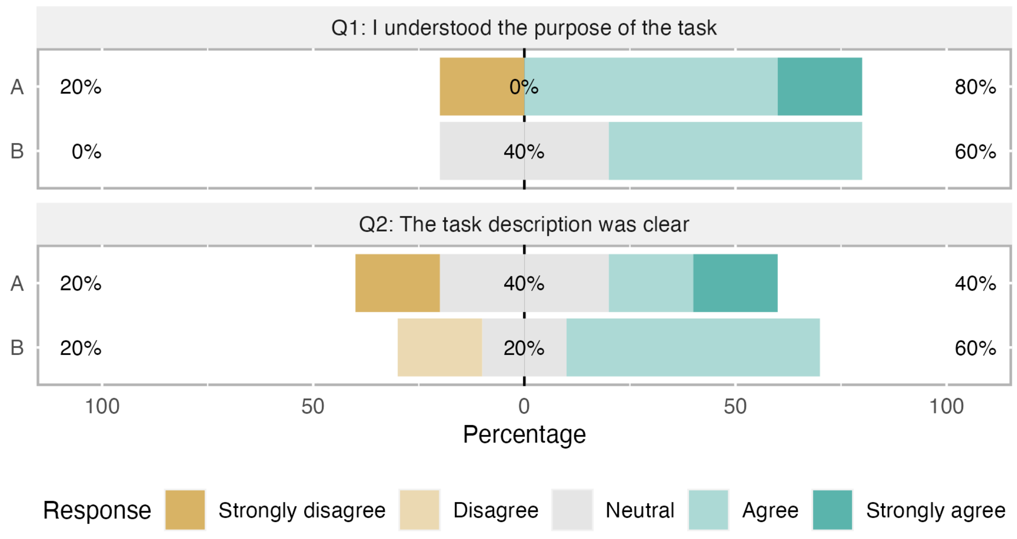


Figure 5.7: Charts of the participants' understanding of the tasks performed. The percentage represents aggregated disagreement, neutrality, and agreement.

To evaluate how well participants understood the task they were performing in the quasi-experiment, we included a section in the survey where the participants self-evaluated their understanding of the tasks. The motivation for this was to identify if any differences in understanding might affect the task execution and survey results. We wrote both task descriptions, and it was important to assess whether they were at the same level.

Overall, the results suggest that participants in both groups generally understood the purpose of the task. However, there were some differences in the perceived clarity of the task descriptions between the two groups. Specifically, participants in task B perceived the task description to be slightly clearer than participants in task A.

Development process

When analyzing the data regarding the development process, the questions were grouped together based on different aspects of the development process:

- the first group, depicted in Figure 5.8, focuses on following the task description to extend immuneML by adding a new machine learning method

- the second group, depicted in Figure 5.9, following the task description to run immuneML with the new machine learning method
- the last group, depicted in Figure 5.10 focuses on the general interpretation of the development process

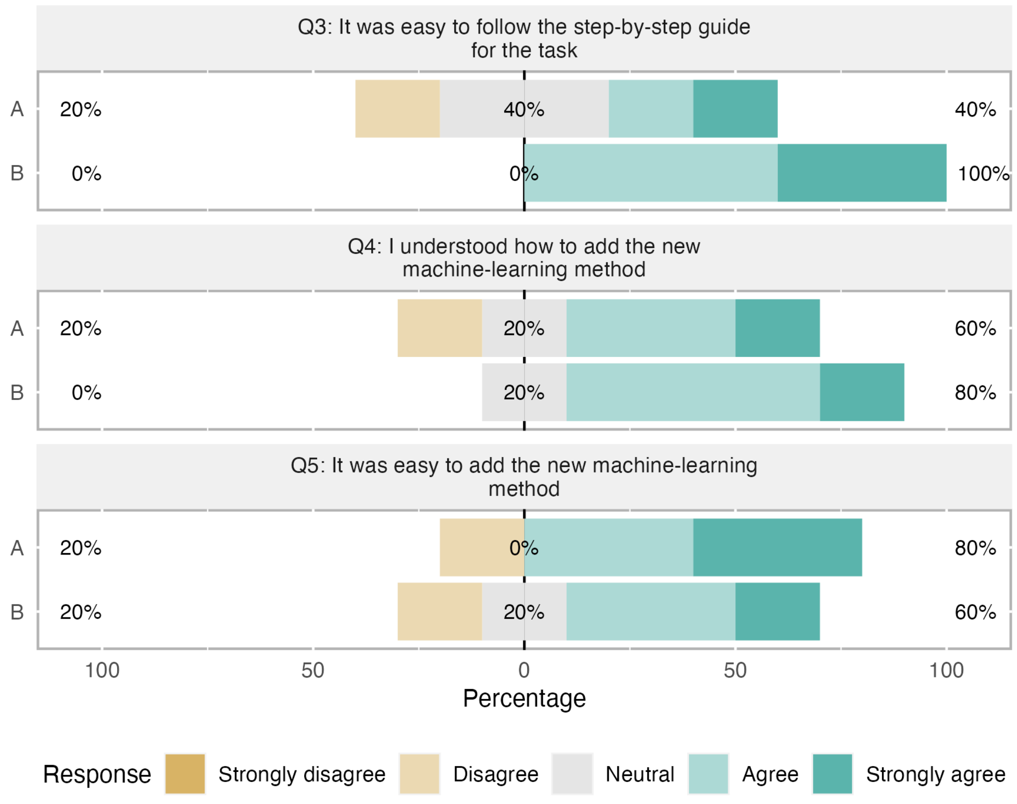


Figure 5.8: Charts of the participants' experience in adding functionality. The percentage represents aggregated disagreement, neutrality, and agreement.

Figure 5.8 depicts the results from how the participants experienced following the task description and extending immuneML with a machine learning method. The results show that the participants performing task B found it easier to follow the guide compared to those performing task A. Both groups understood how to add a machine-learning method. However, there was a higher understandability in task B. On the other hand, the understanding of adding a machine learning method was higher in task A. The overall perception of the model shows that there were no significant differences.

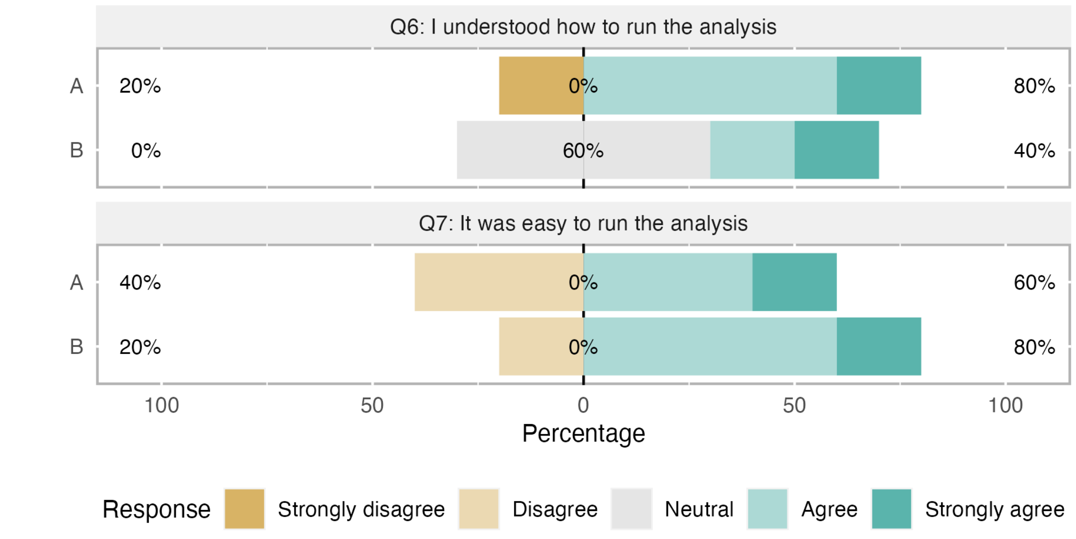


Figure 5.9: Charts of the participants' experience in running immuneML with new functionality. The percentage represents aggregated disagreement, neutrality, and agreement.

Q6, as depicted in Figure 5.9, suggests that participants who performed task A generally had a better understanding of how to run an analysis with the new functionality. As for Q7, which asked about the ease of running the analysis, the group performing task B had a slightly higher number of participants who agreed that it was easy to run the analysis.

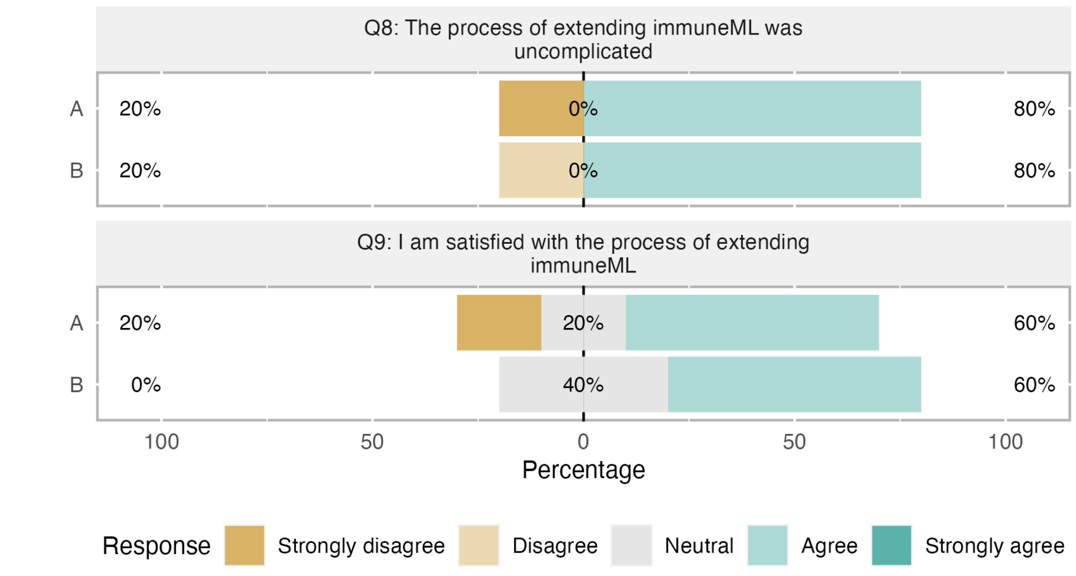


Figure 5.10: Charts of the participants' general experience of the development process. The percentage represents aggregated disagreement, neutrality, and agreement.

The results, shown in Figure 5.10, show how the participants found the overall process of extending immuneML. In Q8, capturing the simplicity of the process shows an even distribution. Similarly, responses to Q9 showing the satisfaction of the process has an even perception.

To summarize the development process, the changes made in the development process did not significantly affect the perception of the development process. Our interpretation is that these results are positive because the process and steps involved in extending the platform in task B are more comprehensive. Overall, the results are similar between the two groups, and there was no significant difference in how the process was perceived.

YAML file

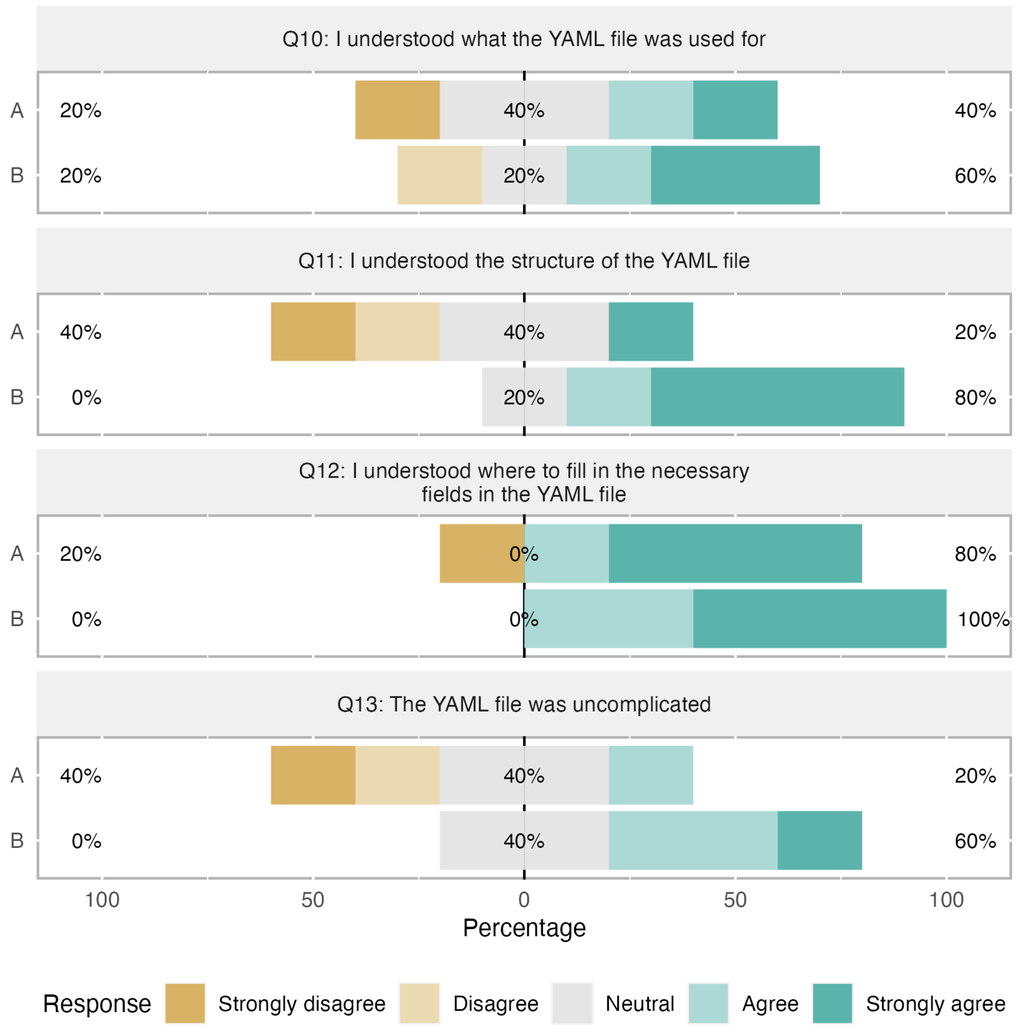


Figure 5.11: Charts of the participants' experience of using the YAML file. The percentage represents aggregated disagreement, neutrality, and agreement.

Based on the data collected from the questionnaire, it was observed that the participants in Task B had a higher understanding of what the YAML file was used for than those in Task A. Additionally, the structure of the YAML file was also better understood by a majority (80%) of the participants in task B as compared to only 20% in task A. Although both groups had a good understanding of where to fill in the YAML file, it was slightly higher in task B.

A noteworthy finding is that the YAML file was perceived as more complex in Task A compared to Task B, even though a new section was added to the YAML file in Task B. This is significant as it suggests that the participants found it easier to comprehend and work with the YAML file in Task B.

In summary, the results show that the understandability of the YAML file was higher in task B. Furthermore, the complexity of the YAML file was perceived to be lower when the participants had to define tools. Our interpretation of the data suggests that increasing the number of steps in the YAML file in task B did not make it more complex or reduce its understandability. Overall, the new process introduced for task B was perceived to be easier to work with and understand.

Response from open-ended questions in questionnaire

Question 1: Were there any parts of extending immuneML that was confusing or difficult?

No significant insights were gained from the question, except for the participants in both Task A and B found the use of immuneML somewhat confusing since they had no prior experience with it.

Question 2: Were there any parts of working with the YAML specification file that was confusing or difficult?

A common theme for both tasks was that the description of how to fill in the YAML file made it relatively easy.

One participant performing task A gave feedback that the instructions were explicit, making it easy to follow, but they felt that understanding how to work with the YAML for other purposes would have posed a challenge.

In Task B, one participant noted that it was *"pretty clear"*, while another mentioned that they were initially confused about where to specify the tool name in the definition and instruction sections. They suggested that there could have been more descriptions to help them with this particular aspect.

Question 3: Do you have any other comments on how you interpreted the process of extending immuneML?

Most participants had "No" as an answer to this question.

In Task A, one of the answers was that the extending immuneML was quite straightforward, but highlighted that it *"... would be costly if multiple ML methods are integrated through this approach"*. Another participant also noted that it was simple to make mistakes following this approach.

In Task B, one participant stated that the process of extending immuneML was simple to follow but that they would have preferred to have a bit more in detail in the task description.

Summary

The quasi-experiment found that participants in both groups generally understood the purpose of the task. Still, there were some differences in the perceived clarity of the task descriptions between the two groups. The changes made to the development process did not significantly affect the perception of the process. The understandability of the YAML file was higher in task B, and the complexity of the YAML file was perceived to be lower when participants had to define tools. Overall, the new process introduced for task B was perceived to be easier to work with and understand.

The summary of introducing the artifact in the process of extending the platform for third-party developers is as follows:

- **Understandability:** The artifact did not have any significant impact on the understandability.
- **Ease of use:** The development process is perceived the same while filling out the YAML file was perceived as more simple.
- **Satisfaction:** The artifact did not have any significant impact on satisfaction.

5.3.2 Data collection 2: Survey with tasks

In this section, we will present the findings of our data collection, which involved users performing tasks and providing feedback via a survey regarding their interpretation of the process. We have organized the results into three distinct categories, similar to our quasi-experiment: understanding of the task, the process, and the YAML file.

Experience of participants

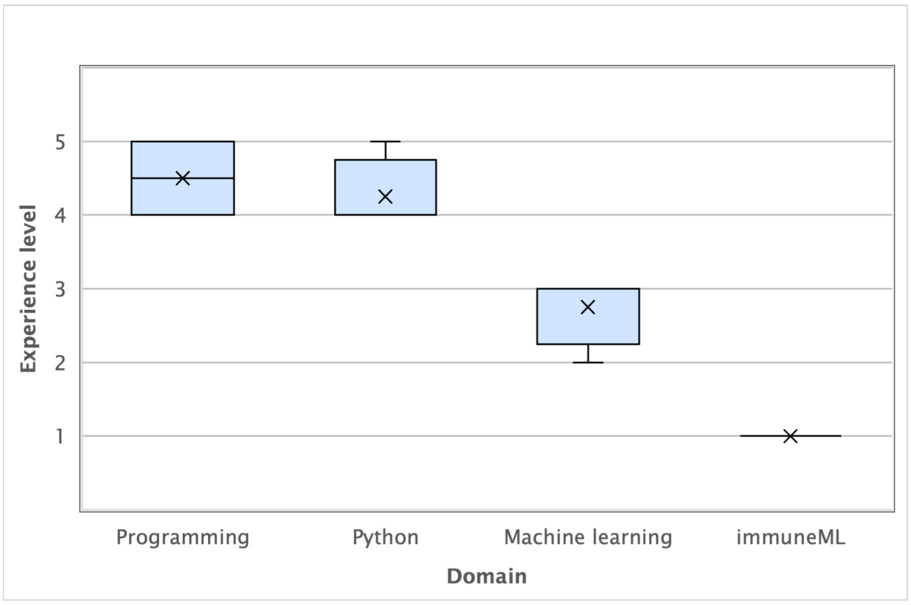


Figure 5.12: Participants' self-assessment of experience. Scale: No experience (1), beginner (2), some experience (3), intermediate (4), advanced (5)

Figure 5.12 shows the participants' background experience. The majority of participants reported proficiency in programming in general and in Python specifically, as well as some proficiency in machine learning. None of the participants had prior experience or knowledge of immuneML.

Understanding of the task

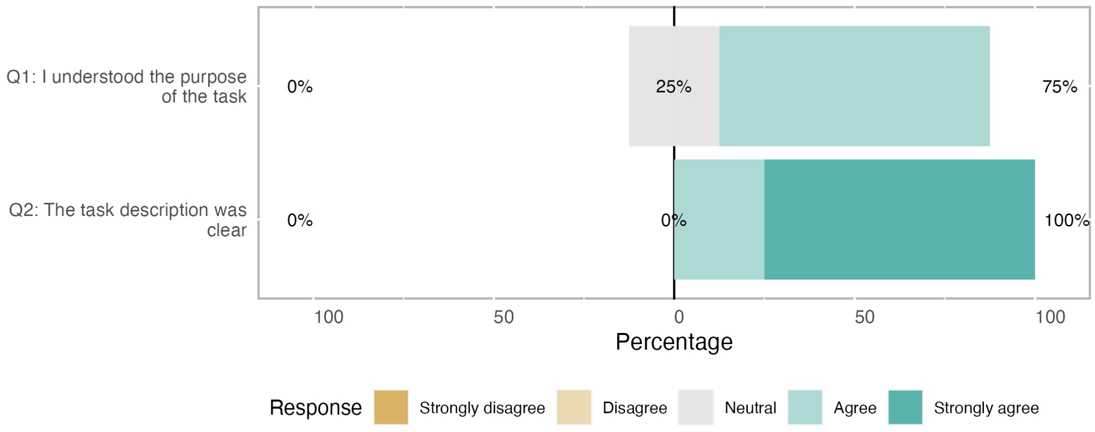


Figure 5.13: Charts of the participants' understanding of the tasks performed. The percentage represents aggregated disagreement, neutrality, and agreement.

The participants had a good understanding of the purpose of the task, with 75% indicating that they understood it well. All of the participants found the task description to be clear, resulting in 100% agreement.

Process

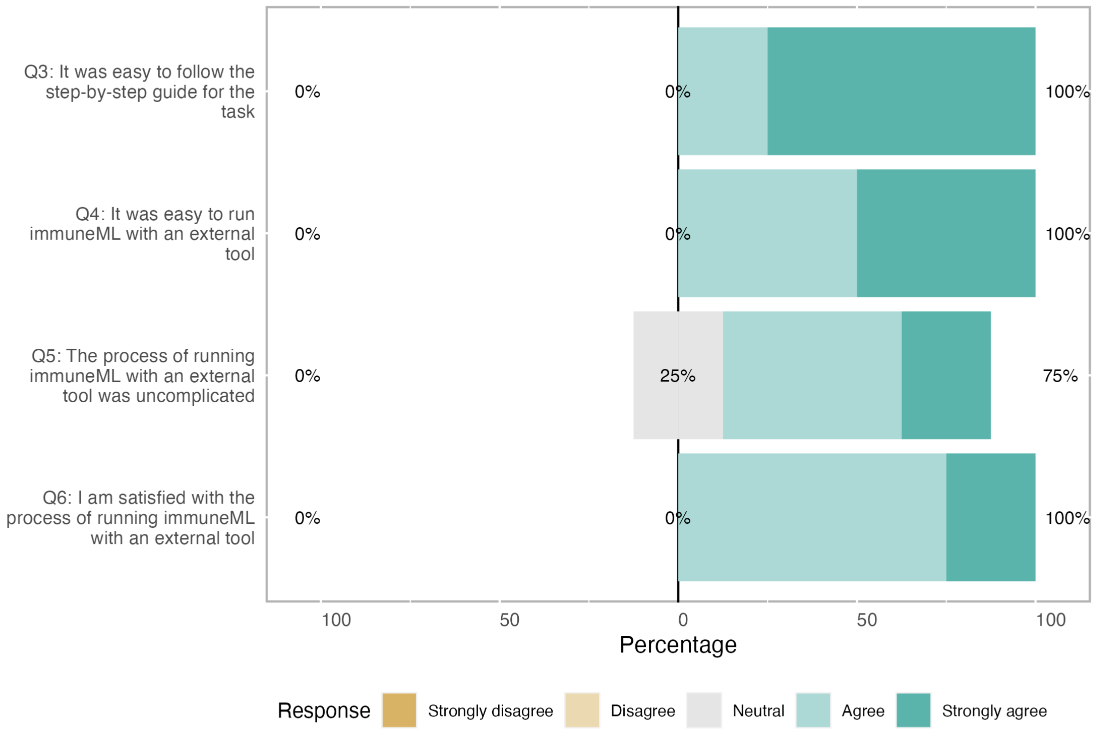


Figure 5.14: Charts of the participants' experience in the process of using immuneML with a tool. The percentage represents aggregated disagreement, neutrality, and agreement.

The participants had a positive experience overall with the process of using immuneML with tools. All participants found the step-by-step guide easy to follow, and all of them found it easy to run immuneML with an external tool. When asked about the complexity of the process, one participant neither agreed nor disagreed, resulting in 75% finding it uncomplicated. The participants' satisfaction was high, with 100% indicating they were satisfied with the process.

YAML

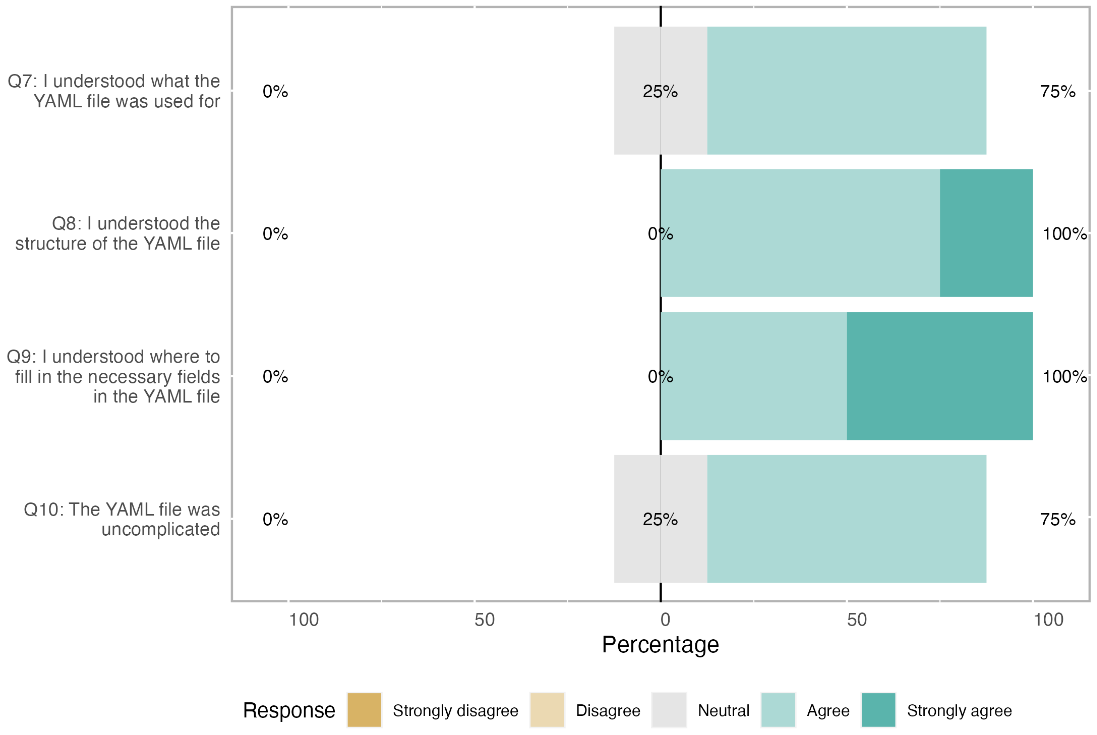


Figure 5.15: Charts of the participants' experience of using the YAML file. The percentage represents aggregated disagreement, neutrality, and agreement.

Finally, in the YAML file section, the participants showed a good understanding of the YAML file, with a score of 75% agreement. Furthermore, they had a very high understanding of the structure of the YAML file and where to fill in the necessary fields, scoring 100% agreement. The majority of participants found the YAML file to be uncomplicated, indicating that the process was not unnecessarily complex.

Responses from open-ended questionnaire

We asked three open-ended questions in the post-task questionnaire:

- **Question 1:** Were there any parts of the process that was confusing or difficult?
- **Question 2:** Were there any parts of working with the YAML specification file that was confusing or difficult?
- **Question 3:** Do you have any other comments on how you interpreted using an external tool with immuneML?

The only question that gave us any feedback was question 1. One participant noted that they were not entirely sure why there was a "tools" section defined in the YAML when they still had to reference it in both the "definitions" and "instructions" sections of the YAML file.

Summary of results

In summary, the participants demonstrated a good understanding of the task, found the process easy to follow, and had a positive overview of the YAML file. Despite one participant's uncertainty about the complexity of the process, overall, the participants found it uncomplicated, and their satisfaction with the process was high.

The summary of the effectiveness of using the artifact to run an external tool written in a different programming language is as follows:

- **Understandability:** there was a high degree of understandability for all participants
- **Ease of use:** all participants found it simple
- **Satisfaction:** all participants found the process satisfactory

5.3.3 Data collection 3: Semi-structured interview

In this section, we present the results from the semi-structured interview with members of the immuneML team. We present the results in five sections: feedback on the process, YAML file, architecture, our solution for communication, and the use of data types. In our presentation of this data, we refer to the final artifact presented in Chapter 6.

Process

Introducing the discussion of our proposed process for third-party developers, we presented both a descriptive model, shown in Figure 4.1, and a prescriptive model, shown in Figure 5.1. The feedback was generally positive, with one participant expressing that the more people are not required to go through the whole code, the better. However, they also noted that while the prescriptive process model seemed reasonable, it might be more complex than the figure depicts because the developer would still need to investigate the interface documentation.

Additionally, we discussed the process of using external tools with immuneML from a user perspective, as illustrated in Figure 5.2. Our solution introduces a new step, where the user has to identify the path for the connection script of the tool and add it to the YAML file. We, therefore, asked if these steps were realistic for a user to do. They responded that it sounded reasonable and that this should all be fine for them. They added that from their perspective, users such as bioinformaticians are often used to working with advanced setups and technical issues, making the process changes easy to adopt.

YAML

The immuneML team provided positive feedback on the changes made to the YAML file structure, with one participant stating, *"I think it looks very nice"* and another agreeing that the fact that it's shorter makes it seem cleaner and easier to relate to. The importance of clear naming was also emphasized. It was suggested that using a different name than "tools" might make it clearer that the tools are running externally and not being imported. However, they also acknowledged that as long as the concept is well-explained to the user, using "tools" is sufficient.

Architecture

When discussing the artifact implementation of the final artifact, described in Chapter 6, one of the participants questioned why we decided to use a controller instead of directly calling tools from the classes in the core. After further discussion, it was demonstrated that using the controller enabled the reuse and encapsulation of the logic related to running tools. The overall impression was that the participants were relatively neutral towards the architecture, neither agreeing nor disagreeing on whether this was a good solution.

Communication between immuneML and external tools

When describing how the logic of the communication in the artifact worked, described in Section 6.4, one of the participants pointed out that this was an interesting solution. However, they questioned if the use of sockets would add some complexity and if it was not simpler to instead rely on the command line. Similarly to the feedback on the architecture, the participants were generally neutral to this solution.

Data types

They also mentioned that TSV files were a good solution, instead of Pickle with the internal data structure in immuneML, as it was not a viable solution for something outside the project. However, they mentioned that TSV files could be too big. Overall, they believed that TSV files with encoded data were a decent strategy.

Summary

In summary, the participants had a positive view of the new process enabled by the artifact. Their opinion on the architecture and the communication method between immuneML and external tools was neutral. However, they were positive towards our decisions regarding data types.

Chapter 6

Final artifact

This chapter will describe the artifact. First, the high-level concepts of the artifact will be described. Secondly, we will describe the artifact's implementation in terms of its architecture and technical details. The artifact developed is a functional prototype designed to meet the immuneML team's requirements and investigate extensibility solutions for ML platforms. Throughout the design and development process, our primary focus was on quality attributes, specifically usability, maintainability, and interoperability, to address the research questions.

6.1 General solution

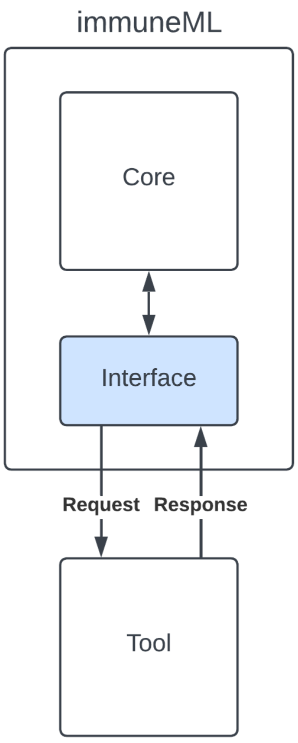


Figure 6.1: High-level model of the general solution.

The interface acts as a boundary between the core (the immuneML source code) and tools and handles all communication and coordination. The high-level concept is illustrated in Figure 6.1. The interface is part of the immuneML platform and is accessible from the core. When an analysis is defined to use a tool, immuneML sends a request to the interface. The interface logic handles the request and forwards it to the tool outside the platform. The tool processes the request and returns the results to the interface, which then sends it back to the immuneML process.

A tool is an executable or script that has implemented the required functionality to connect to immuneML. The interface will manage and execute the tool as a child process by using the Python subprocess module. Communication between the platform and tools is achieved through the ZeroMQ messaging library, which utilizes sockets for inter-process communication.

6.1.1 Changes to the platform

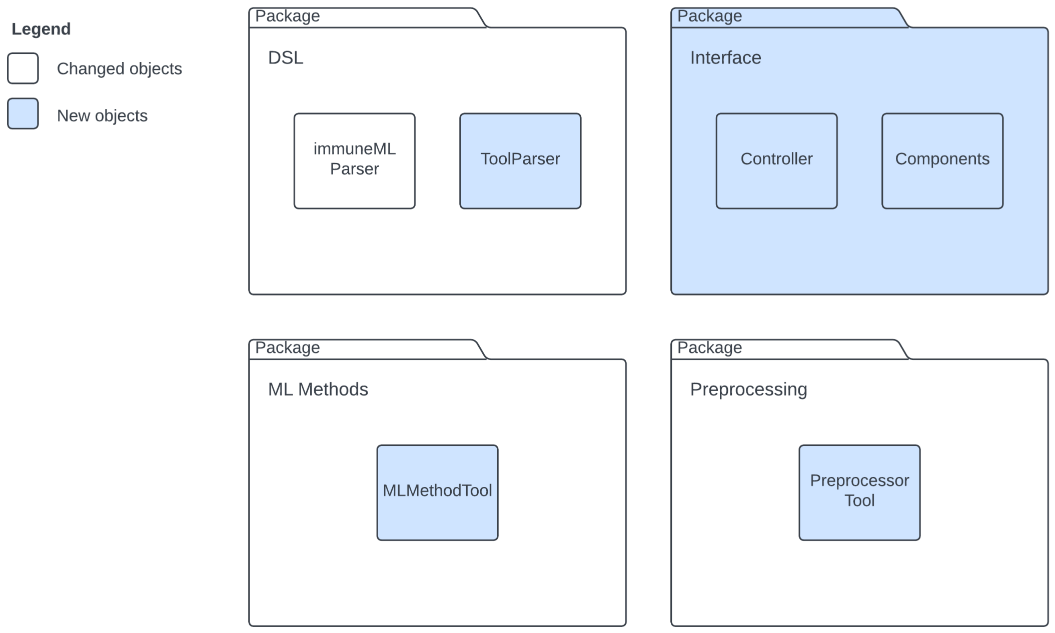


Figure 6.2: UML diagram showing packages and classes changed in artifact

During the development of the artifact, our focus has been to minimize any modifications made to the core. We achieved this by separating the interface from the rest of the core and by using the abstract classes that facilitate the platform's existing extensibility mechanisms. This approach ensures that any modifications made to the interface have minimal impact on the core. Figure 6.2 illustrates the primary changes and additions to the platform. The upcoming sections in this chapter will further explain these changes and classes.

6.2 Interface package

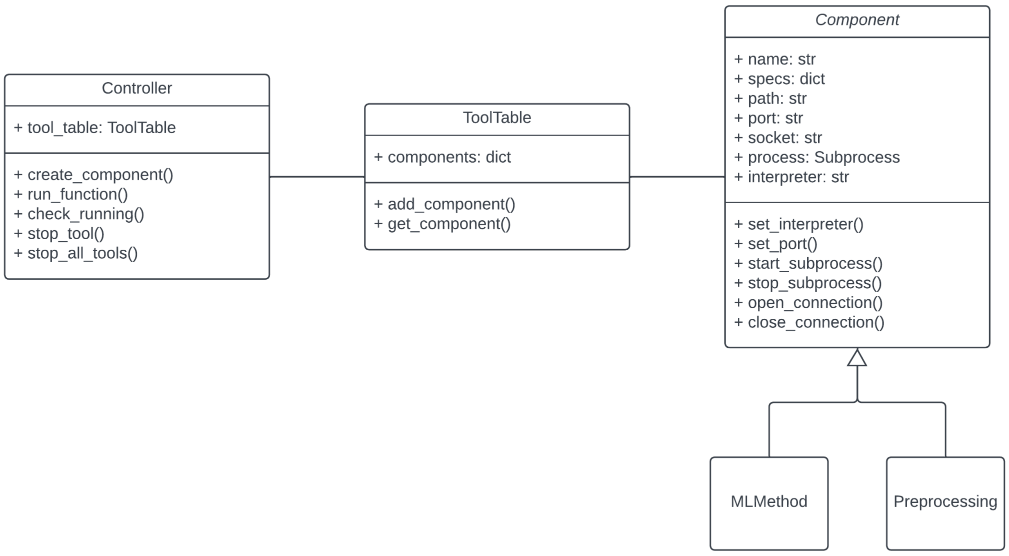


Figure 6.3: UML diagram showing Interface package

The main logic of the tool interface is located in a designated package, illustrated in Figure 6.3. This enforces a boundary between the interface and the rest of immuneML, minimizing dependency and increasing maintainability. The package consists of components for each tool type, a ToolTable, and a controller to manage everything. In this section, we will explain each class of the package.

6.2.1 Tool components

Each tool will have its instance of a tool component. The tool component class contains all the necessary attributes and functions for running and managing the tool and its subprocess. Attributes such as path, socket, and port number, and functions for starting and stopping the subprocess. By gathering all tool-related code in a single class, we are reusing and encapsulating the functionality and data related to running and managing tools.

The tool component is a base class, which each type of tool will inherit to implement the functionality needed for that tool usage. E.g., the ML method has a "fit" function, but the preprocessor does not. The current types of tools supported in this artifact are ML methods (tools for training ML models) and preprocessing (tools used for editing datasets).

6.2.2 Tool table

The ToolTable contains all instances of tool components used in an analysis (parsed by the ToolParser). Each component is added with its identification name as its key, which the controller will use to identify and access the desired component. Gathering all tool instances also enables performing multiple operations on all tools simultaneously, such as making sure that all subprocesses for tools are stopped at the end of an analysis.

6.2.3 Interface controller

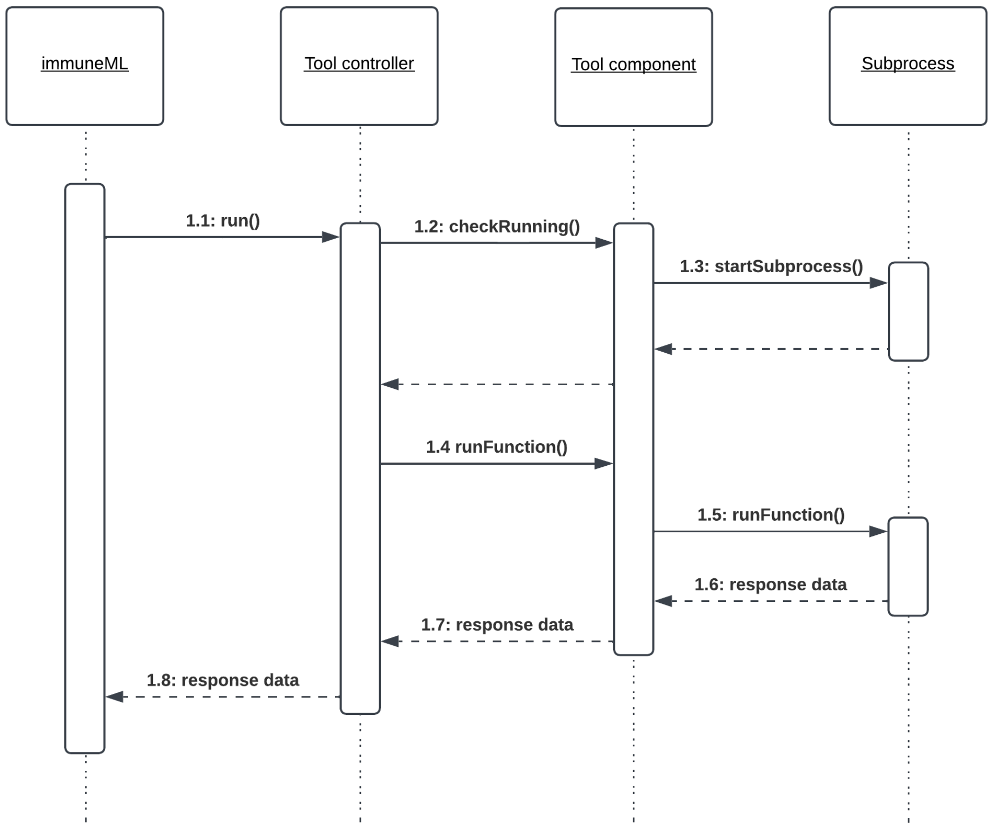


Figure 6.4: UML sequence diagram of running function in tool

The interface controller manages all tool components and communication between the core and the tool. By the use of the ToolTable it is able to store all tools in the same place so they are accessible when the controller is called from the core. Since all tool components inherit from the base class, the controller can treat them similarly, except for tool type-specific functions. (Polymorphism)

The interface controller serves as the only interaction point for the core to the tool interface. Whenever the core needs to access or communicate with the tools, it will send requests to this controller. Figure 6.4 shows an example of an interaction where the immuneML core makes a request through the controller. The "run"-function is used to call tool type-specific functions in tool components, identifying the function to be executed by its name. This approach is adaptable as the functions in the tool components differ depending on their type. Whenever a function in the tool is called, the controller confirms that the tool component has an associated subprocess running. If there is no subprocess, the controller will start a subprocess and initiate communication. This will further be explained in Section 6.4.

6.3 Tool parser

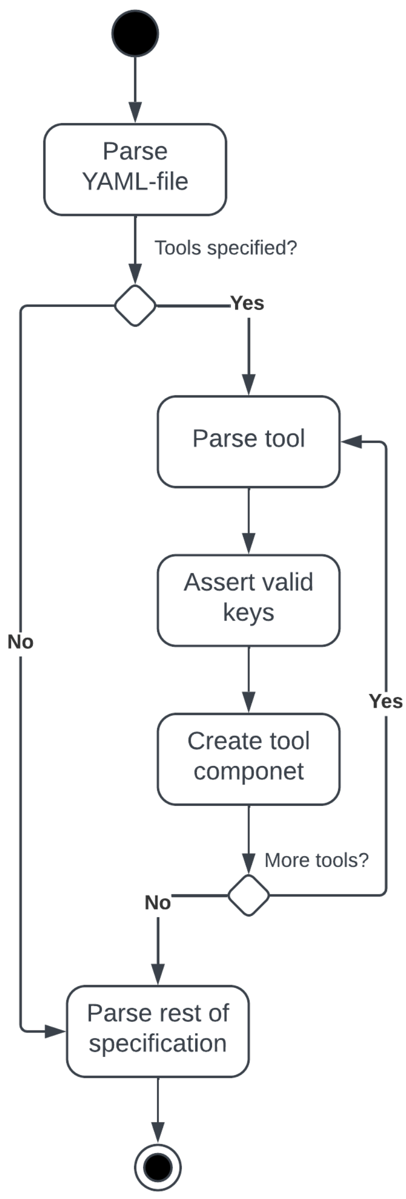


Figure 6.5: UML activity diagram showing flow of the ToolParser

The ToolParser class parses the tool section of the YAML file. It is built similarly to the other parsers in immuneML, where each key that contains one tool will go through the same process. First, we check if all keys present are of a valid type. Second, we check that the required keys are present. If these checks do not pass, the program will exit with an error message to the user explaining what is incorrect and guiding them on what caused the error. If all checks are approved, the interface controller will be called to create a new component with the specified type. When creating a new component, it will be based on the specified path set if it is an interpreter or an executable.

6.4 Communication

The communication logic between immuneML and an external tool is through Inter-Process Communication. While several mechanisms can be used in IPC, we created communication through sockets. Sockets are used for communication between processes running on the same machine but can also be used on different machines over a network. Currently, our artifact only supports usage on a local machine but facilitates further development for usage on different machines. This can be advantageous, e.g., if platform users train on large datasets and need more computational power than their local machine can provide. However, this would require a change to how data is shared as currently the data sharing is based on file sharing.

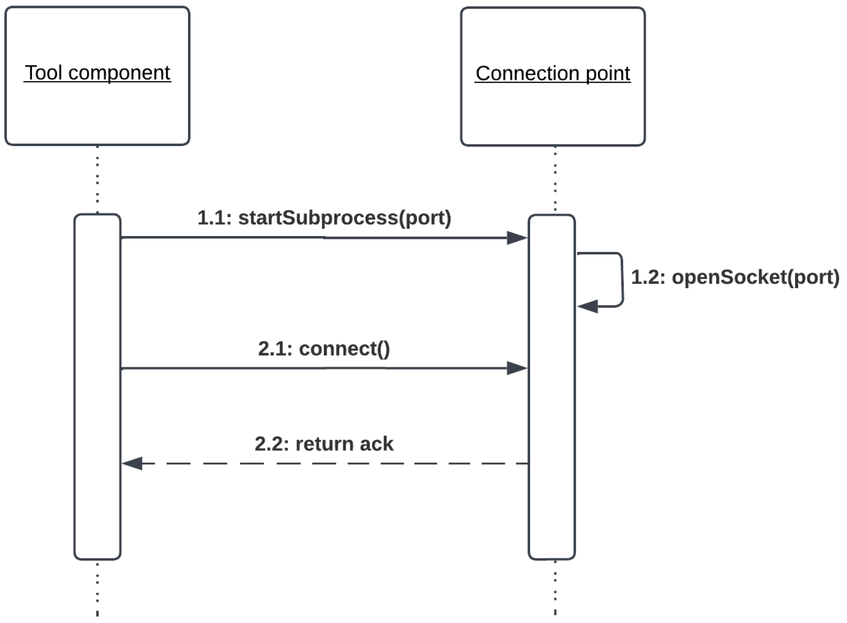


Figure 6.6: UML sequence diagram of initializing tool and communication

To make sure the process for a developer to connect to immuneML is simple, we provide connection blueprints in Python, Java, and C++. The blueprint must be included in the tool to connect it to immuneML and will serve as the interface for the tool for connection. We have named this the connection point. New blueprints must be created if the interface is extended to support additional programming languages. One of the significant benefits of ZeroMQ is its extensive support of programming languages. In other words, the same structure would be followed, and the code itself would only have to be adapted to the functions and syntax of that specific programming language. Examples of the blueprints can be seen in Appendix I.

The process of starting a tool and opening communication is illustrated in Figure 6.6. First, the tool component will find an available port to be used for communication. The tool component will then spawn a child process for the tool and provide the port number through a pipe established by the subprocess Python module. The logic in the connection blueprint will then open a socket using ZeroMQ and work as a service that replies to requests. The tool component will try to connect to the tool and wait until the connection is achieved and the tool returns an acknowledgment message. immuneML and the interface can now send commands on what to run. These messages use the JSON format.

6.4.1 Data types

The interface facilitates tools written in other programming languages than Python by the use of standard data formats. JSON is used for communication between the tool components and the tools. JSON is widely supported, lightweight, and easy to parse. The messages contain information for the tools on what they should run and where to find the datasets by the path. Since this artifact is used to run locally on a computer, we are sending paths to the datasets. The dataset files are in the TSV format. By using standard and widely supported data formats, we are increasing interoperability.

6.4.2 Messaging library - ZeroMQ

For managing communication between the core and tool, we use ZeroMQ, described in Section 2.6.2. Using this messaging library enables the solution to achieve a higher level of interoperability. The ZeroMQ's socket API provides multiple messaging patterns [14]. In our implementation, we use the request-reply pattern, which relies on synchronous communication. The immuneML acts as the client, while the tools act as services. To enable this communication, we utilize two types of ZeroMQ sockets: REQ and REP. With the REQ socket, immuneML can send requests to the tools and receive replies, while the tools use the REP socket to receive requests and send replies back to immuneML.

Chapter 7

Discussion

In this chapter, we will discuss our research questions and present our key findings. We will then highlight the implications of our work for research and practice, and discuss the role of machine learning in shaping our proposed solution. We will also present related work and assess the quality of our research by discussing the validity of our methods and results. Furthermore, we will outline the Design Science Research (DSR) guidelines we followed throughout our study, and discuss how we followed these. We will then discuss the limitations of our study and provide recommendations for future research that can build upon our findings.

7.1 RQ1: How can extending a machine learning platform be improved through an interface?

This study aimed to address the challenges of extending a machine learning platform, specifically focusing on extensibility. Our primary contributions include the design and development of an interface that simplifies the process of extending immuneML, facilitates the integration of tools written in additional programming languages, and enables third-party developers to extend immuneML without requiring in-depth knowledge of the platform. The main research question we aimed to answer in this thesis is:

> *RQ1: How can extending a machine learning platform be improved through an interface?*

To improve the extensibility of immuneML, we explored solutions for how a machine learning platform can be extended in a simple manner and facilitate the external code to be in additional programming languages. Therefore the focus of our work was also on usability and interoperability. We changed the extensibility mechanism by encapsulating the core, enabling third-party developers to extend the platform without being dependent on the immuneML team. We proved the efficacy of our solution by demonstrating that it provides a black-box extensibility mechanism and is a working solution in our use cases.

7.1.1 RQ1.1: How can extending a machine learning platform be simplified through an interface?

We investigated how extending a machine learning platform can be simplified through an interface. Our approach focused on minimal setup for developers to connect new and existing tools to immuneML. Through the use of the messaging library ZeroMQ, we enabled what we believe is a simple and efficient way of establishing communication between immuneML and external tools. Instructions and data were sent in JSON format, giving developers high freedom to use the data fitting their needs.

Through the quasi-experiment, we measured the usability of the solution by investigating how third-party developers perceived the changes in the development process of extending the platform. The task performed by the developers did not include coding and understanding the code but rather following the steps in the development process using the interface. The results, as discussed in Section 3.9.3, showed that the artifact did not significantly impact the understandability and satisfaction of the process. However, since the solution with the interface included more comprehensive steps and managed to improve the extensibility, we perceive this as a positive result. Additionally, the YAML file was found easier to fill in, contributing to positive results of our solution. Overall, all developers successfully extended immuneML using the interface, indicating that it is an effective approach.

Furthermore, the comparison of the descriptive and prescriptive process models for extending the platform for third-party developers, as depicted in Figure 4.1 and Figure 5.1, indicates a reduction in the number of steps required. The new process design simplifies the developer's tasks, resulting in a more efficient and streamlined approach. However, as pointed out by one of the participants in the semi-structured interview discussed in Section 5.3.3, while the third-party developer no longer needs to understand the source code of the platform, they do need to comprehend the interface. Nevertheless, the immuneML team also emphasized that the more people who do not need to go through the source code, the better.

7.1.2 RQ1.2: How can an interface facilitate the integration of tools written in additional programming languages?

To facilitate the integration of tools written in other programming languages, we focused on creating an interface with a high level of interoperability. We accomplished this by using sockets and ZeroMQ, which have bindings in most of the popular programming languages, such as C, C++, Java, and Python. Additionally, to achieve syntactical interoperability, we used common data types. The JSON format was used for passing messages consisting of instructions and file paths, enabling the use of datasets files in the TSV format.

Through our use case of Absolut, we demonstrated that our interface could be used for running programs written in other programming languages, in this case, C++. As a proof-of-concept, it demonstrated how our artifact was able to improve the extensibility of immuneML by introducing functionality that was not previously present, opening up a wider array of tools that can be used with immuneML. We showed through the demonstration of Absolut that the tool was able to connect to immuneML by using a connection script based on the common blueprint.

We further showed through a combination of performing a task and filling in a survey that users were successfully able to download and use Absolut together with immuneML. This was used to simulate and measure the process of using an external tool. The results, discussed in Section 5.3.2, showed that the participants found it understandable, simple, and satisfactory.

7.2 Contributions

The following list summarizes the contributions of our artifact:

- An interface that introduces a new extensibility mechanism to immuneML, going from glass-box to black-box - developers no longer need insights into the source code
- An interface that improves the extensibility of immuneML by simplifying the process
- An interface that improves the extensibility of immuneML by enabling the use of external tools written in additional programming languages
- An architecture that is designed to support immuneML developers in further development of the interface's functionality

7.3 Implications

7.3.1 Implications for research

- The findings of our study highlight the potential benefits of using our developed interface for extending a machine-learning platform. Future research can build on our results by investigating the effectiveness of the interface in more complex use cases and data collections with larger sample sizes.

- Our solution can serve as a starting point for further research on solutions for extending machine learning platforms. Future research could focus on optimizing the performance of our artifact, evaluating its scalability, or comparing it with other approaches.
- While we found that using JSON for messages and TSV files for datasets was a relatively simple and practical approach for our purpose, future research could investigate the use of other data formats and communication for different types of machine-learning platforms.
- The development of our artifact was based on the implementation of immuneML. Future research could investigate the generalizability of our interface to other machine-learning platforms and evaluate its effectiveness in different contexts.

7.3.2 Implications for practice

- Our solution can be used as a reference or basis for similar platforms to enable the development of external tools without working directly with the source code.
- The use of sockets, ZeroMQ, and common data types can be used to increase interoperability. By adopting the same approach, a platform can run tools in different programming languages.
- Introducing an interface can help transition from a glass-box mechanism to a black-box mechanism, thereby increasing the extensibility of a system. This can be useful in practice to allow for a simplified integration of external tools and technologies.

7.4 The impact of ML in shaping the artifact

We have worked with a platform in the context of an extensible codebase, following the platform definition described in Section 2.1.1. Specifically, we examined a platform that is extended via GitHub. In this section, we explore how working with a machine-learning platform has influenced our artifact and reflect on how the outcome might have differed if we had worked with a non-machine-learning platform. It's worth noting that the variances between different types of platforms are significant, so our observations are based on our speculations.

One major challenge is the large size of data files, data sets, and trained models, which can make it difficult to transfer data over the network. Since the platform is designed to run locally on a computer, data can be sent through paths to files, which simplifies the process of handling large data. If we were not working with machine learning and large files, we could have relied on communication through the network.

Various tools employ different data structures; for example, the Scikit-learn framework uses data stored as Numpy arrays, while PyTorch uses Tensors. A step in preparing data for machine learning training is to convert it from its original format, such as text files, into the type supported by the framework. To address this issue, we are sending data as text files in the TSV format, enabling tool developers to convert the data to their specific needs. In contrast, non-machine learning platforms may have more standardized data types, such as data stored in a database and transmitted in JSON format.

The process of running an analysis from start to finish, step-by-step, is another factor that shaped our solution, for instance, from preprocessing and encoding to training. Since these processes are carried out sequentially, we have developed a solution where the tool only needs to run when it is used in each step. When a step starts, the tool initiates, and when it completes, the tool stops because it is no longer necessary for that particular analysis. Consequently, we are only running one tool at a time. In a non-machine learning platform, we anticipate the need for multiple tools to execute through the entire process, enabling the core to make requests to them at any point.

7.5 Related work

Extensibility

In our literature review, we could not locate any specific studies focusing on improving the extensibility of platforms. The closest literature we found that was relevant to our study was the paper by Zenger [57], giving a thorough introduction to different extensibility mechanisms that can be used.

Cross-language interoperability

In the context of enabling the use of multiple programming languages between different systems and programs, we found multiple studies providing a more technical view on enabling cross-language interoperability. While the research papers are more distant from the focus of our research questions, more specifically RQ 1.2, they provide interesting insights into how cross-language interoperability can be achieved.

Bonnal et al. [6] compares several approaches to using software written in different programming languages together, measuring the throughput of the different approaches.

Aleksyuk and Itsykson [3] researched how to create a cross-language integration method that facilitates the use of software components written in different programming languages while minimizing the need for a manual effort by developers.

Grimmer et al. [17] analyzes different approaches for enabling cross-language interoperability by creating a virtual machine that can run different programming languages, composing them in a seamless way.

7.6 Quality of our research

This section will examine the quality of our design science research. Wohlin et al. [55] presents several classification schemes of validity that have been used for case studies and controlled experiments in software engineering. In this section, we will discuss construct validity, internal validity, external validity, conclusion validity, and reliability.

7.6.1 Validity

Validity in a study indicates the trustworthiness of the results [55]. It also denotes to what extent the results are true and not biased by the subjective point of view of the researcher. As a concept, it is subtler [39]. It refers to how closely we think we measure matches what we want to measure.

7.6.2 Construct validity

According to Wieringa [54], construct validity is *"... the degree to which an application of constructs to phenomena is warranted with respect to the research goals and questions"*. It is an aspect of validity that reflects to what extent the studied operational measures represent what the researcher had in mind and what is investigated according to the research questions [55]. It is concerned with the connection between observation and theory.

Interview

A threat to construct validity is when constructs discussed in an interview are not interpreted in the same way between the researcher and the person(s) being interviewed [55]. To reduce this potential threat, we explained the concepts and presented relevant models before asking the interview questions. Furthermore, we actively encouraged the participants to ask questions during our explanations to ensure their understanding of the concepts.

Quasi-experiment and survey with task

One threat to construct validity in this context is that the entire simulation may not accurately reflect the actual activities in the process. This can lead to the results not necessarily presenting how a third-party developer thinks of the development process because they did not code. To mitigate this threat, we communicated to the participants that the task was to show the process, and we reduced the threat further by having the same kind of tasks and descriptions for both tasks.

Another threat to the construct validity of our results is the potential for questions in the survey to be interpreted differently than what we intended, leading to misunderstandings. To mitigate this threat, we based our questions on examples provided by Tullis and Albert [46] that focused on self-evaluation in post-task questionnaires related to user experience. We also conducted a pilot test. The result of the pilot test helped us improve unclear questions.

A potential threat to construct validity in our quasi-experiment is the possibility that the concepts we aimed to capture may not have been accurately represented in the questions we used. Our goal was to evaluate the usability of the artifact by assessing participants' user experience with regard to factors such as understandability, ease of use, and satisfaction. If our questions did not effectively capture these underlying constructs, the results might not provide an accurate reflection of participants' actual perceptions and experiences. To mitigate this threat, we connected the constructs to the questions using the examples by Tullis and Albert [46]

7.6.3 Internal validity

Threats to internal validity look at issues that may indicate that there is a causal relationship, but in reality, there is none [55]. In other words, when observing a relationship between the treatment and outcome, it is important to verify that a casual relationship exists and not a result of uncontrollable or unmeasured factors [55].

A potential threat to the internal validity of our quasi-experiment was the risk of uneven distribution of experience levels. This would make it difficult to compare the results and give reliable results. To reduce the risk of such an uneven distribution, we divided the participants into two different groups based on assumptions of their skill levels. However, this introduced selection bias, which according to Kampenes et al. [28], is considered a threat to internal validity.

7.6.4 External validity

External validity concerns the concept of generalization [55]. Threats to external validity refer to conditions that limit the ability of the results to be generalized.

Kampenes et al. [28] states that external validity can suffer because of the artificial setting of the quasi-experiment, which can be very different from the real-life setting where the artifact is to be used. Thus, the artificial setting of our quasi-experiment is a threat to external validity.

The external validity of our quasi-experiment is threatened by the small sample size and limited scope of the population used, which may limit the generalizability of our results. We addressed this threat by having a more heterogenous group, meaning that we included participants with different backgrounds, as described in Section 3.9.3.

We have only tested our artifact on one specific platform, immuneML, which it was developed for. In other words, the artifact as a whole can not be generalized to other platforms due to the unique internal structure of immuneML.

Looking at smaller details of the artifact, the generalizability can be addressed differently. The platform currently employs a glass-box extensibility mechanism achieved through the use of abstract classes. Our solution extends this approach by utilizing abstract classes to ensure that modifications to the system do not impact its overall functionality. Therefore, other software systems that utilize a similar extensibility mechanism may find our architectural solution to be useful. Additionally, similar platforms that run locally on the computer may use the findings of how we initialize and communicate with external tools.

7.6.5 Conclusion validity

Threats to conclusion validity refer to issues affecting the ability to draw the correct conclusions about the relations between the treatment and outcome of experiments [55].

While utilizing descriptive analysis, we recognized patterns based on our interpretation. However, because of the small sample size, we cannot conduct statistical tests to confirm and conclude our findings. Consequently, there is a greater probability of drawing inaccurate conclusions.

To reduce the threat to conclusion validity, we attempted to get a more heterogenous group of participants, referring to the background of the participants (i.e., Ph.D., students, and developers).

7.6.6 Reliability

The concept of reliability refers to the extent to which data and analysis depend on a researcher [55]. If the same study were repeated by different researchers, the results should hypothetically be the same. Reliability is used for our qualitative data and is the counterpart to conclusion validity that is used for quantitative data [55].

To ensure reliability, we collected data using different methods: a quasi-experiment, a survey with tasks, a semi-structured interview, and a focus group to evaluate the interface. Additionally, we used a consistent methodology for collecting the data.

The reliability of qualitative data may be a concern when it relies solely on the analysis and interpretation of the researchers. To reduce this threat, we coded the data independently and then discussed our findings.

The reliability of the quantitative data may be reduced when the analysis is solely based on our subjective interpretation. To mitigate this, we have provided all the charts used to analyze the data, enabling other researchers to interpret independently.

7.7 Research guidelines

Design Science Research is a problem-solving process [20]. According to Hevner et al. [20], the fundamental principle of Design Science Research is that the process of building and applying an artifact is the way researchers acquire knowledge and understanding of a design problem and its solution. Hevner et al. [20] presents seven guidelines derived from this very fundamental principle to assist researchers and others in better understanding the requirements for what would be considered to be effective Design Science Research. While Hevner et al. [20] advises against mandatory/rote use of the guidelines, they argue that each should be addressed in some form for a Design Science Research to be considered complete. This section will discuss how we followed these guidelines in our research project.

Guideline 1: Design an Artifact

"Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation" [20].

The guideline for design as an artifact includes creating a novel and useful artifact, which in our case, is an interface that allows the use of external tools with the immuneML platform. The artifact's novelty is that it makes it possible to run tools in different programming languages previously not possible in the immuneML platform. Its usefulness is demonstrated through a focus group, a semi-structured interview, a quasi-experiment, and a survey with tasks, which showed positive results regarding the extensibility of the immuneML platform.

Guideline 2: Problem Relevance

"The objective of design-science research is to develop technology-based solutions to important and relevant business problems" [20].

Our thesis addressed the limitations of extensibility in the immuneML platform. By designing an interface that overcomes this limitation, we demonstrated how it could run external code with immuneML and increase the platform's extensibility. This contribution has the potential to benefit users of immuneML by enabling them to use immuneML with a broader range of external tools and facilitating the integration of new ones.

Guideline 3: Design Evaluation

"The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods" [20].

We utilized a focus group, an interview, a quasi-experiment, and a survey with tasks to evaluate our interface's usefulness in facilitating the use of external tools with the immuneML platform. The results demonstrated that the interface created a new platform extension process and enabled the use of external tools written in different programming languages together with immuneML.

Guideline 4: Research Contributions

"Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies" [20].

We believe our thesis contributed to the field of software engineering by demonstrating how an interface can be created to increase extensibility and enable running external code with a platform that previously did not support it. Furthermore, we demonstrated how our interface could enable the use of external tools written in other programming languages, potentially benefiting users by enabling the use of new tools with immuneML that previously would not be possible.

Guideline 5: Research Rigor

"Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact" [20].

Following this guideline, we conducted a literature review and designed evaluations following established scientific principles. To ensure the reliability and consistency of our results, we conducted thematic and descriptive analysis, which provided insight into our data. This allowed us to draw conclusions from our findings and establish the validity of our research contributions.

Guideline 6: Design as a Search Process

"The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment" [20].

We followed this guideline by iterating on the interface design based on feedback from the focus group and incorporating new features to improve its design and functionality.

Guideline 7: Communication of Research

"Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences" [20].

Our thesis communicates our research findings through our presentation of our artifact in this thesis.

7.8 Limitations

Our study has several limitations that should be acknowledged. In this section, we explain the limitations in the context of participants, data analysis, and the size of the conducted quasi-experiment.

7.8.1 Participants

One limitation of our study was the relatively small number of participants we recruited for our summative naturalistic ex post evaluation. Due to the resource-demanding nature of this evaluation strategy, we faced challenges in reaching out to a sufficient number of potential participants.

7.8.2 Analysis

The reason for conducting a descriptive analysis was mainly based on the size of our data points. As a result of few data points, meaning few participants, we did not have enough data to conduct statistical analysis. We attempted to identify any correlations between the results and the participants' experience but were unable to find any.

7.8.3 Size of quasi-experiment

Another limitation we encountered was the constrained nature of the tasks in our quasi-experiment. We had to carefully consider the time and resource constraints involved in designing our evaluation. Ideally, we would have preferred to simulate the full development process by having participants work with the connection script. However, this would have been time-consuming and challenging to recruit participants. As a result, we opted for a scaled-down quasi-experiment that did not include testing the use of the connection script.

Summary of limitations

Overall, our study's limitations should be considered when interpreting the results and generalizing the findings. Future studies could address these limitations by expanding the number of participants and tasks included and exploring additional strategies for recruitment and evaluation.

7.9 Final thoughts

By using the design science research method, we have gained a deeper understanding of the research process, from defining the problem to developing a working artifact. The iterative process allowed us to refine our approach and adapt to challenges that emerged during the project.

Throughout the research, we faced several obstacles, including time constraints and limited resources. However, we produced a solution that met our objectives by maintaining a systematic approach, following the design science research methodology, and focusing on the key research questions.

We acknowledge that our study has limitations and potential areas for improvement. For instance, our quasi-experiment and survey with tasks were conducted in a controlled environment, and future studies could consider a more diverse sample to increase generalizability. Additionally, the study could have included more rigorous evaluation methods to assess the solution's effectiveness.

Overall, the design science research method provided us with a framework to approach our research question systematically and allowed us to develop an artifact that addressed our research questions.

7.10 Future work

One of the main limitations of our study is that we did not examine how external developers would experience following the pattern/blueprint for creating the script necessary to connect the tool together with immuneML. In other words, we did not examine the process to its full extent. To address this limitation, future work should focus on conducting a more extensive evaluation that includes a larger number of participants who are external developers. This would provide a more comprehensive understanding of how developers experience following the pattern/blueprint and how the solution could be improved to better meet their needs.

Another area for further research could be to extend the type of participants beyond just developers. In our quasi-experiment, we primarily focused on participants who were responsible for copying code and filling in a YAML file. While this approach was effective for our purposes, it may not reflect the experiences of other potential users. Thus, future work should investigate how the solution could be adapted to meet the needs of other users, such as bioinformaticians.

Additionally, our research used Inter-Process Communication with the messaging library ZeroMQ to connect external tools with immuneML. While we experienced this as appropriate for our purposes, there is a possibility that other technologies may be more suitable for this purpose in different contexts. Therefore, future work could explore alternative technologies and evaluate their usability and simplicity compared to our solution.

Lastly, adaptations would have to be made for further work using the solution with immuneML to increase its usability. This could include addressing any technical issues that may arise during the use of the interface, such as errors.

Chapter 8

Conclusion

Our research project aimed to develop an interface that allows external tools to be used with the machine-learning platform immuneML. The current immuneML platform does not have a solution to integrate external tools without directly developing into its GitHub repository, which limits the platform's extensibility. Following the design science research methodology, we created an alternative way of extending immuneML without the need to work directly with the source code. Through this interface, we enabled the use of external tools and tools written in different programming languages.

To evaluate our interface, we conducted evaluations consisting of a focus group, a quasi-experiment, a survey with tasks, and a semi-structured interview. In the focus group, we gathered feedback from the platform developers on the interface's usability and potential use cases. We then performed a quasi-experiment to evaluate the effectiveness of the interface. Lastly, we interviewed members of the immuneML team to gain final feedback on our developed solution.

Our research contribution demonstrates the value of creating an interface to increase the extensibility of a platform. By creating this interface for immuneML, we have shown an approach to improve extensibility through a solution focusing on simplicity and interoperability. We believe this work can potentially encourage further research in this area and contribute to the development of more extensible platforms in the future.

Bibliography

[1] *Absolut!* en. July 2020. URL: https://greifflab.org/absolut/ (visited on 08/05/2023).

[2] *Absolut!* original-date: 2020-10-07T08:59:39Z. Mar. 2023. URL: https://github.com/csi-greifflab/Absolut (visited on 31/03/2023).

[3] Artyom Aleksyuk and Vladimir Itsykson. 'Automated Cross-Language Integration Based on Formal Model of Components'. en. In: *Frontiers in Software Engineering Education*. Ed. by Jean-Michel Bruel et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 357–370. ISBN: 978-3-030-57663-9. DOI: 10.1007/978-3-030-57663-9_23.

[4] Len Bass, Paul Clements and Rick Kazman. *Software Architecture in Practice, 4th Edition*. en. 4th Edition. Addison-Wesley Professional, Aug. 2021. ISBN: 978-0-13-688597-9. URL: https://learning.oreilly.com/library/view/software-architecture-in/9780136885979/ (visited on 25/03/2023).

[5] Len Bass, Paul Clements and Rick Kazman. *Software Architecture in Practice, Third Edition*. en. Third Edition. Addison-Wesley Professional, Sept. 2012. ISBN: 978-0-13-294279-9. URL: https://learning.oreilly.com/library/view/software-architecture-in/9780132942799/ (visited on 12/05/2023).

[6] Raoul J. P. Bonnal et al. 'Sharing Programming Resources Between Bio* Projects'. In: *Methods in molecular biology (Clifton, N.J.)* 1910 (Jan. 2019), pp. 747–766. ISSN: 1064-3745. DOI: 10.1007/978-1-4939-9074-0_25. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7212028/ (visited on 12/02/2023).

[7] Virginia Braun and Victoria Clarke. 'Thematic analysis'. In: *APA handbook of research methods in psychology, Vol 2: Research designs: Quantitative, qualitative, neuropsychological, and biological*. APA handbooks in psychology®. Washington, DC, US: American Psychological Association, 2012, pp. 57–71. ISBN: 978-1-4338-1005-3. DOI: 10.1037/13620-004.

[8] Virginia Braun and Victoria Clarke. 'Using thematic analysis in psychology'. In: *Qualitative Research in Psychology* 3.2 (Jan. 2006), pp. 77–101. ISSN: 1478-0887. DOI: 10.1191/1478088706qp063oa. URL:

https://www.tandfonline.com/doi/abs/10.1191/1478088706qp063oa (visited on 14/05/2023).

[9] Jan vom Brocke, Alan Hevner and Alexander Maedche. 'Introduction to Design Science Research'. en. In: *Design Science Research. Cases*. Ed. by Jan vom Brocke, Alan Hevner and Alexander Maedche. Progress in IS. Cham: Springer International Publishing, 2020, pp. 1–13. ISBN: 978-3-030-46781-4. DOI: 10.1007/978-3-030-46781-4_1. URL: https://doi.org/10.1007/978-3-030-46781-4_1 (visited on 07/09/2022).

[10] Tolga Can. 'Introduction to Bioinformatics'. en. In: *miRNomics: MicroRNA Biology and Computational Analysis*. Ed. by Malik Yousef and Jens Allmer. Methods in Molecular Biology. Totowa, NJ: Humana Press, 2014, pp. 51–71. ISBN: 978-1-62703-748-8. DOI: 10.1007/978-1-62703-748-8_4. URL: https://doi.org/10.1007/978-1-62703-748-8_4 (visited on 31/03/2023).

[11] Mikaela Cashman et al. 'Navigating the maze: the impact of configurability in bioinformatics software'. In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ASE '18. New York, NY, USA: Association for Computing Machinery, Sept. 2018, pp. 757–767. ISBN: 978-1-4503-5937-5. DOI: 10.1145/3238147.3240466. URL: https://dl.acm.org/doi/10.1145/3238147.3240466 (visited on 05/05/2023).

[12] Victoria Clarke and Virginia Braun. 'Thematic analysis'. In: *The Journal of Positive Psychology* 12.3 (May 2017). Publisher: Routledge _eprint: https://doi.org/10.1080/17439760.2016.1262613, pp. 297–298. ISSN: 1743-9760. DOI: 10.1080/17439760.2016.1262613. URL: https://doi.org/10.1080/17439760.2016.1262613 (visited on 24/04/2023).

[13] Murray J. Fisher and Andrea P. Marshall. 'Understanding descriptive statistics'. en. In: *Australian Critical Care* 22.2 (May 2009), pp. 93–97. ISSN: 10367314. DOI: 10.1016/j.aucc.2008.11.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S1036731408001732 (visited on 13/05/2023).

[14] *Get started*. URL: https://zeromq.org/get-started/ (visited on 05/04/2023).

[15] Ahmad Ghazawneh and Ola Henfridsson. 'A Paradigmatic Analysis of Digital Application Marketplaces'. en. In: *Journal of Information Technology* 30.3 (Sept. 2015). Publisher: SAGE Publications Ltd, pp. 198–208. ISSN: 0268-3962. DOI: 10.1057/jit.2015.16. URL: https://doi.org/10.1057/jit.2015.16 (visited on 04/05/2023).

[16] Ilya Grigorik. *ZeroMQ: Modern & Fast Networking Stack - igvita.com*. en. Sept. 2010. URL: https://www.igvita.com/2010/09/03/zeromq-modern-fast-networking-stack/ (visited on 06/04/2023).

[17] Matthias Grimmer et al. 'Cross-Language Interoperability in a Multi-Language Runtime'. In: *ACM Transactions on Programming Languages and Systems* 40.2 (2018), 8:1–8:43. ISSN: 0164-0925. DOI: 10.1145/3201898. URL: https://dl.acm.org/doi/10.1145/3201898 (visited on 04/05/2023).

[18] K. Henttonen et al. 'Integrability and Extensibility Evaluation from Software Architectural Models – A Case Study'. In: *The Open Software Engineering Journal* 1.1 (Dec. 2007). ISSN: 1874-107X. DOI: 10.2174/1874107X00701010001. URL: https://benthamopen.com/ABSTRACT/TOSEJ-1-1 (visited on 03/05/2023).

[19] Alan Hevner. 'A Three Cycle View of Design Science Research'. In: *Scandinavian Journal of Information Systems* 19 (Jan. 2007).

[20] Alan Hevner et al. 'Design Science in Information Systems Research'. In: *Management Information Systems Quarterly* 28 (Mar. 2004), p. 75.

[21] 'IEEE Standard Glossary of Software Engineering Terminology'. In: *IEEE Std 610.12-1990* (Dec. 1990). Conference Name: IEEE Std 610.12-1990, pp. 1–84. DOI: 10.1109/IEEESTD.1990.101064.

[22] immuneML. *Developer documentation — immuneML 2.2.4 documentation*. URL: https://docs.immuneml.uio.no/latest/developer_docs.html (visited on 17/04/2023).

[23] *IntegratingPythonWithOtherLanguages - Python Wiki*. URL: https://wiki.python.org/moin/IntegratingPythonWithOtherLanguages (visited on 13/05/2023).

[24] Nusrat Jahan et al. 'How to Conduct a Systematic Review: A Narrative Literature Review'. en. In: *Cureus* 8.11 (Nov. 2016). Publisher: Cureus. ISSN: 2168-8184. DOI: 10.7759/cureus.864. URL: https://www.cureus.com/articles/5127-how-to-conduct-a-systematic-review-a-narrative-literature-review (visited on 28/04/2023).

[25] Paul Johannesson and Erik Perjons. *An Introduction to Design Science*. en. Cham: Springer International Publishing, 2014. ISBN: 978-3-319-10631-1 978-3-319-10632-8. DOI: 10.1007/978-3-319-10632-8. URL: http://link.springer.com/10.1007/978-3-319-10632-8 (visited on 12/01/2023).

[26] Manu Joseph. *PyTorch Tabular: A Framework for Deep Learning with Tabular Data*. arXiv:2104.13638 [cs]. Apr. 2021. DOI: 10.48550/arXiv.2104.13638. URL: http://arxiv.org/abs/2104.13638 (visited on 21/03/2023).

[27] Hanna Kallio et al. 'Systematic methodological review: developing a framework for a qualitative semi-structured interview guide'. en. In: *Journal of Advanced Nursing* 72.12 (2016). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/jan.13031, pp. 2954–2965. ISSN: 1365-2648. DOI: 10.1111/jan.13031. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/jan.13031 (visited on 19/04/2023).

[28] Vigdis By Kampenes et al. 'A systematic review of quasi-experiments in software engineering'. en. In: *Information and Software Technology*. Special Section - Most Cited Articles in 2002 and Regular Research Papers 51.1 (Jan. 2009), pp. 71–82. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2008.04.006. URL: https://www.sciencedirect.com/science/article/pii/S0950584908000670 (visited on 12/04/2023).

[29] Parampreet Kaur, Jill Stoltzfus and Vikas Yellapu. 'Descriptive statistics'. en. In: *International Journal of Academic Medicine* 4.1 (2018), p. 60. ISSN: 2455-5568. DOI: 10.4103/IJAM.IJAM_7_18. URL: http://www.ijam-web.org/text.asp?2018/4/1/60/230853 (visited on 13/05/2023).

[30] Felipe da Veiga Leprevost et al. 'On best practices in the development of bioinformatics software'. In: *Frontiers in Genetics* 5 (2014). ISSN: 1664-8021. URL: https://www.frontiersin.org/articles/10.3389/fgene.2014.00199 (visited on 03/05/2023).

[31] *ml-jku/DeepRC: DeepRC: Immune repertoire classification with attention-based deep massive multiple instance learning*. en. URL: https://github.com/ml-jku/DeepRC (visited on 14/05/2023).

[32] Jürgen Münch et al. *Software Process Definition and Management*. The Fraunhofer IESE Series on Software and Systems Engineering. Berlin, Heidelberg: Springer, 2012. ISBN: 978-3-642-24290-8 978-3-642-24291-5. DOI: 10.1007/978-3-642-24291-5. URL: http://link.springer.com/10.1007/978-3-642-24291-5 (visited on 17/04/2023).

[33] Sri Manikanta Palakollu. 'Interprocess Communication'. en. In: *Practical System Programming with C: Pragmatic Example Applications in Linux and Unix-Based Operating Systems*. Ed. by Sri Manikanta Palakollu. Berkeley, CA: Apress, 2021, pp. 165–214. ISBN: 978-1-4842-6321-1. DOI: 10.1007/978-1-4842-6321-1_6. URL: https://doi.org/10.1007/978-1-4842-6321-1_6 (visited on 05/04/2023).

[34] Milena Pavlović et al. 'The immuneML ecosystem for machine learning analysis of adaptive immune receptor repertoires'. en. In: *Nature Machine Intelligence* 3.11 (Nov. 2021). Number: 11 Publisher: Nature Publishing Group, pp. 936–944. ISSN: 2522-5839. DOI: 10.1038/s42256-021-00413-z. URL: https://www.nature.com/articles/s42256-021-00413-z (visited on 09/05/2023).

[35] Ken Peffers et al. 'A Design Science Research Methodology for Information Systems Research'. In: *Journal of Management Information Systems* 24.3 (Dec. 2007). Publisher: Routledge _eprint: https://doi.org/10.2753/MIS0742-1222240302, pp. 45–77. ISSN: 0742-1222. DOI: 10.2753/MIS0742-1222240302. URL: https://doi.org/10.2753/MIS0742-1222240302 (visited on 13/05/2023).

[36] Julie Ponto. 'Understanding and Evaluating Survey Research'. In: *Journal of the Advanced Practitioner in Oncology* 6.2 (2015), pp. 168–171. ISSN: 2150-0878. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4601897/ (visited on 28/04/2023).

[37] Jan Pries-Heje, Richard Baskerville and John Venable. 'Strategies for Design Science Research Evaluation'. In: *ECIS 2008 Proceedings* (Jan. 2008). URL: https://aisel.aisnet.org/ecis2008/87.

[38] Mark de Reuver, Carsten Sørensen and Rahul C. Basole. 'The Digital Platform: A Research Agenda'. In: *Journal of Information Technology* 33.2 (June 2018). Publisher: SAGE Publications Ltd, pp. 124–135. ISSN: 0268-3962. DOI: 10.1057/s41265-016-0033-3. URL: https://doi.org/10.1057/s41265-016-0033-3 (visited on 04/05/2023).

[39] Paula Roberts and Helena Priest. 'Reliability and validity in research'. In: *Nursing standard (Royal College of Nursing (Great Britain) : 1987)* 20 (July 2006), pp. 41–5. DOI: 10.7748/ns2006.07.20.44.41.c6560.

[40] S Roopa and MS Rani. 'Questionnaire Designing for a Survey'. In: *Journal of Indian Orthodontic Society* 46.4_suppl1 (Oct. 2012). Publisher: SAGE Publications India, pp. 273–277. ISSN: 0301-5742. DOI: 10.5005/jp-journals-10021-1104. URL: https://journals.sagepub.com/doi/abs/10.5005/jp-journals-10021-1104 (visited on 26/03/2023).

[41] David Saffo, Laura South and Amy Worth. 'Evaluating Visualization Styles for Likert Scale Data'. en-us. In: (Nov. 2020). Publisher: OSF. DOI: 10.17605/OSF.IO/5R28V. URL: https://osf.io/5r28v (visited on 14/05/2023).

[42] Paul T. Shannon et al. 'The Gaggle: An open-source software system for integrating bioinformatics software and data sources'. In: *BMC Bioinformatics* 7.1 (Mar. 2006), p. 176. ISSN: 1471-2105. DOI: 10.1186/1471-2105-7-176. URL: https://doi.org/10.1186/1471-2105-7-176 (visited on 03/05/2023).

[43] Abdulhamit Subasi. *Chapter 3: Machine learning techniques*. en. ISBN: 978-0-12-821380-3. URL: https://learning.oreilly.com/library/view/practical-machine-learning/9780128213803/xhtml/B9780128213797000035.xhtml (visited on 14/04/2023).

[44] *subprocess — Subprocess management*. URL: https://docs.python.org/3/library/subprocess.html (visited on 09/05/2023).

[45] *TensorFlow*. URL: https://www.tensorflow.org/ (visited on 21/03/2023).

[46] Tom Tullis and Bill Albert. *Measuring the User Experience*. English. Interactive Technologies. https://doi.org/10.1016/C2011-0-00016-9. Morgan Kaufmann, 2013. ISBN: 978-0-12-415781-1.

[47] John Venable. 'A framework for design science research activities'. In: *Proceedings of the 2006 Information Resource Management Association Conference* (Jan. 2006), pp. 184–187.

[48] John Venable, Jan Pries-Heje and Richard Baskerville. 'A Comprehensive Framework for Evaluation in Design Science Research'. en. In: *Design Science Research in Information Systems. Advances in Theory and Practice*. Ed. by Ken Peffers, Marcus Rothenberger and Bill Kuechler. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 423–438. ISBN: 978-3-642-29863-9. DOI: 10.1007/978-3-642-29863-9_31.

[49] John Venable, Jan Pries-Heje and Richard Baskerville. 'FEDS: a Framework for Evaluation in Design Science Research'. en. In: *European Journal of Information Systems* 25.1 (Jan. 2016), pp. 77–89. ISSN: 1476-9344. DOI: 10.1057/ejis.2014.36. URL: https://doi.org/10.1057/ejis.2014.36 (visited on 06/01/2023).

[50] Aditya Venkataraman and Kishore Kumar Jagadeesha. 'Evaluation of Inter-Process Communication Mechanisms'. en. In: *Architecture* 86 (), p. 64. URL: https://pages.cs.wisc.edu/~adityav/Evaluation_of_Inter_Process_Communication_Mechanisms.pdf.

[51] *Welcome to the immuneML documentation! — immuneML 2.2.4 documentation*. URL: https://docs.immuneml.uio.no/latest/index.html (visited on 12/05/2023).

[52] *What is a Container? | Docker*. en-US. Nov. 2021. URL: https://www.docker.com/resources/what-container/ (visited on 09/05/2023).

[53] Michael Widrich et al. *Modern Hopfield Networks and Attention for Immune Repertoire Classification*. arXiv:2007.13505 [cs, q-bio, stat]. July 2020. URL: http://arxiv.org/abs/2007.13505 (visited on 12/05/2023).

[54] Roel J. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. en. Berlin, Heidelberg: Springer, 2014. ISBN: 978-3-662-43838-1 978-3-662-43839-8. DOI: 10.1007/978-3-662-43839-8. URL: https://link.springer.com/10.1007/978-3-662-43839-8 (visited on 14/05/2023).

[55] Claes Wohlin et al. *Experimentation in Software Engineering*. en. Berlin, Heidelberg: Springer, 2012. ISBN: 978-3-642-29043-5 978-3-642-29044-2. DOI: 10.1007/978-3-642-29044-2. URL: http://link.springer.com/10.1007/978-3-642-29044-2 (visited on 06/04/2023).

[56] *YAML specification — immuneML 2.2.4 documentation*. URL: https://docs.immuneml.uio.no/latest/specification.html (visited on 12/05/2023).

[57] Matthias Zenger. 'Programming language abstractions for extensible software components'. eng. PhD thesis. Lausanne: EPFL, 2004. DOI: 10.5075/epfl-thesis-2930.

[58] *ZeroMQ*. URL: https://zeromq.org/ (visited on 05/04/2023).

[59] Zhi-Hua Zhou. *Machine Learning*. en. Singapore: Springer, 2021. ISBN: 9789811519666 9789811519673. DOI: 10.1007/978-981-15-1967-3. URL: https://link.springer.com/10.1007/978-981-15-1967-3 (visited on 06/05/2023).

Appendix A

Github repository

Repository for thesis project

https://github.com/osk10/immuneML

Repository for use case 1, Absolut

https://github.com/jskimmeland/Absolut

Repository for use case 2, DeepRC

https://github.com/osk10/DeepRC-usecase

Appendix B

Consent form

Consent form for an interview regarding master thesis project

Contact information

Oskar Lund	+47 944 81 537	oskarl@ifi.uio.no
Jørgen Skimmeland	+47 900 92 090	jorgeski@ifi.uio.no

Description of the master thesis project

We are two students from the Institute for Informatics at the Faculty of Mathematics and Natural Sciences, University of Oslo. Under the research group Software Engineering, we are writing a master thesis about tool integration for the machine learning platform immuneML. Our supervisors are Antonio Martini and Karthik Shivashankar.

Through the development of an artifact, the project aims to improve the extensibility of a machine-learning platform.

We wish to invite you to participate in this interview. The goal of the interview is to present our developed artifact and collect data based on your feedback.

Participation

Participation is voluntary, and you can at any time withdraw from the interview. For our data collection, we wish to take notes of the interview. The notes will only be reviewed by us and deleted by 01.06.2023. The participants will be referred to as members of the immuneML team, not by names.

Consent

I have read and understood the information above, and I agree to participate in this interview

________________ ________________________________

Place and date Signature

Appendix C

Quasi-experiment Survey Task A

Nettskjema

The form should be anonymous. **Show more**

immuneML tasks

0 %

Mandatory fields are marked with a star *

Introduction

Our master’s thesis project involves improving the process of extending the immuneML platform, a platform used for machine learning based analysis and classification of adaptive immune receptors and repertoires. As part of our research, we would like to invite you to participate in performing a task and answer a survey. This will be used to evaluate how individuals experience the development process of extending the immuneML platform on a small scale.

This survey consists of three parts:

1. A questionnaire to gather some information about your experience with programming, immuneML, and machine learning.
2. A task to extend the immuneML platform.
3. A questionnaire that asks about your experience in performing the task.

You should not spend more than 1 hour in total, and there is no requirement for you to have finished the task. We recommend you have 10 minutes to answer the final questionnaire.

Your participation in this survey is highly appreciated, and all responses will be kept strictly confidential. The survey responses will be deleted by 1. June 2023.

Next page

Responsible for the form:
oskarl@uio.no

Nettskjema

The form should be anonymous. **Show more**

immuneML tasks

20 %

Mandatory fields are marked with a star *

Participant information

What is your task? *

Before this survey, you should have been informed of which task you will solve.

- ◯ Task A
- ◯ Task B

What is your experience in programming? *

- ◯ No experience
- ◯ Beginner
- ◯ Some experience
- ◯ Intermediate
- ◯ Advanced

What is your experience in Python? *

- ◯ No experience
- ◯ Beginner
- ◯ Some experience
- ◯ Intermediate
- ◯ Advanced

What is your experience in machine learning? *

- ◯ No experience
- ◯ Beginner
- ◯ Some experience
- ◯ Intermediate
- ◯ Advanced

What is your experience with immuneML? *

- ○ No experience
- ○ Beginner
- ○ Some experience
- ○ Intermediate
- ○ Advanced

Previous page

Next page

Responsible for the form:
oskarl@uio.no

Nettskjema

The form should be anonymous. **Show more**

immuneML tasks

40 %

Mandatory fields are marked with a star *

Installment

If you have not installed immuneML we ask you to follow these instructions: https://docs.immuneml.uio.no/latest/developer_docs/install_for_development.html#set-up-immuneml-for-development

Task preparation

Before you can start the task you will have to download some resources.

1. Download zip: https://drive.google.com/file/d/1vZ8AgV-T8uiq3drf32yjqOg6z1mtz2h7/view?usp=share_link
2. Unzip, and place the “task_resources” folder somewhere so you are easily able to find it later.

The folder contains:

- dataset - a folder containing a metadata file and repertoires
- specs.yaml - a YAML specification file
- code_method.txt - file containing code that will be used in the task

Setup finished

The setup is now finished, and you can go ahead with the task. You will later be asked to estimate the time used on the task, and should now start tracking your time.

Previous page

Next page

Responsible for the form:
oskarl@uio.no

The form should be anonymous. **Show more**

immuneML tasks

60 %

Mandatory fields are marked with a star *

Task description

As a developer, you want to use a new machine learning method with immuneML. You already have an implementation of a logistic regression method and want to integrate it with immuneML. In this task, you will extend the immuneML platform by adding the new machine-learning method and run an analysis in immuneML using this method.

Instructions

Part 1 - add a new machine learning method to immuneML

1. Open the immuneML project in PyCharm (or any other code editor).
2. Add a new Python file called “NewLogisticRegression” to the package ml_methods.
3. From the downloaded resources folder: copy the content from the file “code_method.txt”.
   - This file contains the class NewLogisticRegression with an implemented logistic regression ml method using the Scikit-learn framework.
4. Paste the content into the new file “NewLogisticRegression”.

You have now extended immuneML with a new ML method. In the next part, you will run an analysis using this method.

Part 2 - run an analysis in immuneML with your new machine-learning method

Analyses in immuneML are specified through a YAML specification file with a fixed structure of nested key-value pairs. Depending on the specification, immuneML can execute different tasks, such as training ML models. In the resources folder, a partially filled YAML file is available, which needs to be edited to work with your new method.

1. From the downloaded resources folder: Open the file “specs.yaml” in a code editor.
2. The YAML file requires you to fill in both the definitions and instructions sections. Specify the new ML method by editing the unfinished YAML file
   1. In the “definitions”-section: define the method
   2. In the “instructions”-section: add the method to the “my_training_instruction”
   - Documentation on how to fill in a YAML file: https://github.com/osk10/immuneML/blob/dev/YAML-instructions-taskA.md
3. Save the edited file and run the analysis using the following steps:
   1. Through the command line, activate the virtual environment where immuneML is available.
   2. Through the command line, navigate to the downloaded folder “task_resources” where ”specs.yaml" is located.
   3. Run this command: *immune-ml specs.yaml ./analysis_results/*
      - This command will run immuneML with the YAML specification file, and save the analysis results in a folder generated by immuneML
4. You have successfully completed this task when the final output is “ImmuneML: finished analysis.”

You have now extended immuneML with a new machine-learning method. Please take note of your time and go to the next page!

Nettskjema

The form should be anonymous. **Show more**

immuneML tasks

80 %

Mandatory fields are marked with a star *

Task questionnaire

Were you able to finish the task? *

○ Yes

○ No

Approximately how many minutes did you spend on preparation before starting the task? *

Approximately how many minutes did you spend on doing the task? *

Task description

	Strongly disagree	Disagree	Neither agree nor disagree (neutral)	Agree	Strongly agree
I understood the purpose of the task *	○	○	○	○	○
The task description was clear *	○	○	○	○	○

Development process

This section is about how you experienced the process of extending immuneML with a new machine learning method

	Strongly disagree	Disagree	Neither agree nor disagree (neutral)	Agree	Strongly agree
It was easy to follow the step-by-step guide for the task *	○	○	○	○	○
I understood how to add the new machine-learning method *	○	○	○	○	○
It was easy to add the new machine-learning method *	○	○	○	○	○
I understood how to run the analysis *	○	○	○	○	○
It was easy to run the analysis *	○	○	○	○	○
The process of extending immuneML was uncomplicated *	○	○	○	○	○
I am satisfied with the process of extending immuneML *	○	○	○	○	○

YAML file

This section is about how you experienced filling in the YAML file.

	Strongly disagree	Disagree	Neither agree nor disagree (neutral)	Agree	Strongly agree
I understood what the YAML file was used for *	○	○	○	○	○
I understood the structure of the YAML file *	○	○	○	○	○
I understood where to fill in the necessary fields in the YAML file *	○	○	○	○	○
The YAML file was uncomplicated *	○	○	○	○	○

Was there any parts of extending immuneML that was confusing or difficult? *

Was there any parts of working with the YAML specification file that was confusing or difficult? *

Do you have any other comments on how you interpreted the process of extending immuneML? *

Previous page

Send

Responsible for the form:
oskarl@uio.no

Appendix D

Quasi-experiment Survey Task B

Nettskjema

The form should be anonymous. **Show more**

immuneML tasks

0 %

Mandatory fields are marked with a star *

Introduction

Our master's thesis project involves improving the process of extending the immuneML platform, a platform used for machine learning based analysis and classification of adaptive immune receptors and repertoires. As part of our research, we would like to invite you to participate in performing a task and answer a survey. This will be used to evaluate how individuals experience the development process of extending the immuneML platform on a small scale.

This survey consists of three parts:

1. A questionnaire to gather some information about your experience with programming, immuneML, and machine learning.
2. A task to extend the immuneML platform.
3. A questionnaire that asks about your experience in performing the task.

You should not spend more than 1 hour in total, and there is no requirement for you to have finished the task. We recommend you have 10 minutes to answer the final questionnaire.

Your participation in this survey is highly appreciated, and all responses will be kept strictly confidential. The survey responses will be deleted by 1. June 2023.

Next page

Responsible for the form:
oskarl@uio.no

Nettskjema

The form should be anonymous. **Show more**

immuneML tasks

20 %

Mandatory fields are marked with a star *

Participant information

What is your task? *

Before this survey, you should have been informed of which task you will solve.

- ○ Task A
- ○ Task B

What is your experience in programming? *

- ○ No experience
- ○ Beginner
- ○ Some experience
- ○ Intermediate
- ○ Advanced

What is your experience in Python? *

- ○ No experience
- ○ Beginner
- ○ Some experience
- ○ Intermediate
- ○ Advanced

What is your experience in machine learning? *

- ○ No experience
- ○ Beginner
- ○ Some experience
- ○ Intermediate
- ○ Advanced

What is your experience with immuneML? *

○ No experience

○ Beginner

○ Some experience

○ Intermediate

○ Advanced

Previous page

Next page

Responsible for the form:
oskarl@uio.no

The form should be anonymous. **Show more**

immuneML tasks

40 %

Mandatory fields are marked with a star *

Installment

You will be using a new version of immuneML, which you will have to install before starting the task.

Download

1. Download zip file of the project by clicking this link https://github.com/osk10/immuneML/tree/workshop
2. Unzip, and place the folder somewhere so you are easily able to find it later.

Installation

1. Create a new Python virtual environment and activate it.
   - Use your preferred solution for handling virtual environments. Below are resources for setting up virtual environments.
2. Through the command line, navigate to the downloaded immuneML folder.
3. Install immuneML by running these commands:

   pip install -r requirements.txt

   pip install -e .

Virtual environments resources:

Venv

Documentation: https://docs.python.org/3.8/library/venv.html

Conda

Documentation: https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html

1. Create a new virtual environment for the project by running this command. (Replace <env_name> with an optional name)

   conda create -n <env_name> python=3.8

2. Activate the virtual environment by running this command:

   conda activate <env_name>

Task preparation

Before you can start the task you will have to download some resources.

1. Download zip: https://drive.google.com/file/d/1_QKR4IaPlpKbQHifvT168HFRD70uiQxX/view?usp=share_link
2. Unzip, and place the “task_resources” folder somewhere so you are easily able to find it later.

The folder contains:

- dataset - a folder containing a metadata file and repertoires
- specs.yaml - a YAML specification file
- code_method.txt - file containing code that will be used in the task
- code_connection.txt - file containing code that will be used in the task

Setup finished

The setup is now finished, and you can go ahead with the task. You will later be asked to estimate the time used on the task, and should now start tracking your time.

Previous page

Next page

Responsible for the form:
oskarl@uio.no

Nettskjema

The form should be anonymous. **Show more**

immuneML tasks

60 %

Mandatory fields are marked with a star *

Task description

As a developer, you want to use a new machine learning method with immuneML. You already have an implementation of a logistic regression method and want to use it with immuneML. The new version of immuneML has a programming interface, that facilitates the integration of external tools. By using this programming interface, you will extend immuneML with the new machine learning method located outside the platform, and use it as part of an analysis.

Instructions

Part 1 - extend the immuneML platform with a new method tool

1. Create a new folder / Python project that will contain your code
2. Open the folder in PyCharm (or any other code editor).
3. Create a new Python file in the folder/project named “MyMethod”
4. From the downloaded resources folder: Copy the content from the file “code_method.txt”.
5. Paste the content into “MyMethod”.
   1. This file contains an implementation of the ML method logistic regression.
6. To connect your project to immuneML's interface, you have to add a new Python file that will serve as the connection point to immuneML.
   1. Create a new Python file in the folder/project named “main”
   2. From the downloaded resources folder: Copy the content from the file “code_connection.txt”.
      - This file contains an implementation to communicate with the immuneML platform.
   3. Paste the content into “main”

You have now created a ML method tool outside of immuneML with a file to connect with the immuneML platform. In the next part, you will run an analysis with this tool.

Part 2 - run immuneML with your tool

Analyses in immuneML are specified through a YAML specification file with a fixed structure of nested key-value pairs. Depending on the specification, immuneML can execute different tasks, such as training ML models. In the resources folder, a partially filled YAML file is available, which needs to be edited to work with your tool.

1. From the downloaded resources folder: Open the file “specs.yaml” in a code editor.
2. The YAML file requires you to fill in the definitions, instructions and tools-sections. Specify the new ML method by editing the unfinished YAML file
   1. In the "tool"-section: define the tool
   2. In the “definitions”-section: define the method
   3. In the “instructions”-section: add the method to the “my_training_instruction”
   - Documentation on how to fill in a YAML file: https://github.com/osk10/immuneML/blob/dev/YAML-instructions-taskB.md
3. Save the edited file and run the analysis using the following steps:
   1. Through the command line, activate the virtual environment where immuneML is available.
   2. Through the command line, navigate to the downloaded folder “task_resources” where ”specs.yaml" is located.
   3. Run this command: *immune-ml specs.yaml ./analysis_results/*
      - This command will run immuneML with the YAML specification file, and save the analysis results in a folder generated by immuneML
4. You have successfully completed this task when the final output is “ImmuneML: finished analysis.”

You have now extended immuneML with a new machine-learning method tool. Please take note of your time and go to the next page!

Responsible for the form:
oskarl@uio.no

Nettskjema

The form should be anonymous. **Show more**

immuneML tasks

80 %

Mandatory fields are marked with a star *

Task questionnaire

Were you able to finish the task? *

○ Yes

○ No

Approximately how many minutes did you spend on preparation before starting the task? *

Approximately how many minutes did you spend on doing the task? *

Task description

	Strongly disagree	Disagree	Neither agree nor disagree (neutral)	Agree	Strongly agree
I understood the purpose of the task *	○	○	○	○	○
The task description was clear *	○	○	○	○	○

Development process

This section is about how you experienced the process of extending immuneML with a new machine learning method

	Strongly disagree	Disagree	Neither agree nor disagree (neutral)	Agree	Strongly agree
It was easy to follow the step-by-step guide for the task *	○	○	○	○	○
I understood how to add the new machine-learning method *	○	○	○	○	○
It was easy to add the new machine-learning method *	○	○	○	○	○
I understood how to run the analysis *	○	○	○	○	○
It was easy to run the analysis *	○	○	○	○	○
The process of extending immuneML was uncomplicated *	○	○	○	○	○
I am satisfied with the process of extending immuneML *	○	○	○	○	○

YAML file

This section is about how you experienced filling in the YAML file.

	Strongly disagree	Disagree	Neither agree nor disagree (neutral)	Agree	Strongly agree
I understood what the YAML file was used for *	○	○	○	○	○
I understood the structure of the YAML file *	○	○	○	○	○
I understood where to fill in the necessary fields in the YAML file *	○	○	○	○	○
The YAML file was uncomplicated *	○	○	○	○	○

Was there any parts of extending immuneML that was confusing or difficult? *

Was there any parts of working with the YAML specification file that was confusing or difficult? *

Do you have any other comments on how you interpreted the process of extending immuneML? *

Previous page Send

Responsible for the form:
oskarl@uio.no

Appendix E

Absolut Survey

Nettskjema

The form should be anonymous. **Show more**

immuneML Absolut

0 %

Mandatory fields are marked with a star *

Introduction

Our master's thesis project involves how extending the immuneML platform can be improved. immuneML is a platform used for machine learning-based analysis and classification of adaptive immune receptors and repertoires. An essential part of this project is to enable running tools written in different programming languages with immuneML. As part of our research, we would like to invite you to participate in performing a task and answer a survey. This will be used to evaluate how individuals experience using external tools with immuneML.

This survey consists of three parts:

1. A questionnaire to gather some information about your experience with programming, immuneML, and machine learning.
2. A task to run immuneML with an external tool
3. A questionnaire that asks about your experience in performing the task.

You should not spend more than 1 hour in total, and there is no requirement for you to have finished the task. We recommend you have 10 minutes to answer the final questionnaire.

Your participation in this survey is highly appreciated, and all responses will be kept strictly confidential. The survey responses will be deleted by 1. June 2023.

Next page

Responsible for the form:
jorgeski@uio.no

Nettskjema

The form should be anonymous. **Show more**

immuneML Absolut

20 %

Mandatory fields are marked with a star *

Participant information

What is your experience in programming? *

- ○ No experience
- ○ Beginner
- ○ Some experience
- ○ Intermediate
- ○ Advanced

What is your experience in Python? *

- ○ No experience
- ○ Beginner
- ○ Some experience
- ○ Intermediate
- ○ Advanced

What is your experience in machine learning? *

- ○ No experience
- ○ Beginner
- ○ Some experience
- ○ Intermediate
- ○ Advanced

What is your experience with immuneML? *

- ○ No experience
- ○ Beginner
- ○ Some experience
- ○ Intermediate
- ○ Advanced

Previous page

Next page

Responsible for the form:
jorgeski@uio.no

The form should be anonymous. **Show more**

immuneML Absolut

40 %

Mandatory fields are marked with a star *

Installment

In this task, you will be using a modified version of immuneML. This must be downloaded before you can continue with the task.

Download

1. Download the zip file of the project by following this link https://github.com/osk10/immuneML/tree/preprocessing
2. Unzip and place the folder somewhere so you can easily find it later.

Installation

The version of immuneML you have downloaded is directed toward further development. As a result, you will have to follow a few steps to prepare immuneML. To be able to run immuneML, it is required that you use Python version 3.8. immuneML also requires several dependencies to be downloaded. To make sure all of these requirements are satisfied, you will set up a virtual environment following these steps:

1. Create a new Python virtual environment and activate it.
   - Use your preferred solution for handling virtual environments. Below are resources for setting up virtual environments.
2. Through the command line, navigate to the downloaded immuneML folder.
3. Once inside the immuneML folder, install immuneML by running these commands:

   pip install -r requirements.txt

   pip install -e .

Virtual environments resources:

Venv

Documentation: https://docs.python.org/3.8/library/venv.html

Conda

Documentation: https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html

1. Create a new virtual environment for the project by running this command. (Replace <env_name> with an optional name)

   conda create -n <env_name> python=3.8

2. Activate the virtual environment by running this command:

   conda activate <env_name>

Setup finished

The setup is now finished, and you can go ahead with the task. You will later be asked to estimate the time used on the task, and should now start tracking your time.

Previous page

Next page

Responsible for the form:
jorgeski@uio.no

Nettskjema

The form should be anonymous. **Show more**

immuneML Absolut

60 %

Mandatory fields are marked with a star *

Task description

As a user, you want to use immuneML to generate a dataset, preprocess it and export it. To preprocess the dataset, you want to use an external program called Absolut. To do this, you will be using a new version of immuneML with a programming interface that facilitates the integration of external tools. More specifically:

1. You will use a dataset called "RandomSequenceDataset" which is a dataset generated by immuneML that consi-sts of random sequences.
2. Preprocessing in immuneML consists of modifying a dataset before it is further used. You will be using Absolut for this to add additional data to the dataset.
3. To perform these actions with immuneML, you will fill in a YAML specification file. These files are used to specify what analysis immuneML should perform. In other words, they are used to specify what you will do in immuneML

Part 1 - set up Absolut to run with immuneML

To use Absolut to preprocess a dataset, you will first have to download the software and then make it runnable. To download and make Absolut runnable, you will have to follow these steps:

1. Download the zip file called "Absolut.zip" using this link: https://drive.google.com/drive/folders/1qoq0jsLoPgXxs-EUvmP5--tU7wvGqwN9q
2. Unzip the downloaded file and locate it somewhere that is easy to find later
3. Open a terminal on your computer and navigate to where you have saved Absolut
4. To make Absolut runnable, navigate to the folder called "immuneML_interface. This folder contains a file that must be turned into an executable. Follow the navigation below:
    1. Absolut --> Absolut-main --> src --> immuneML_interface
5. Once inside the folder immuneML_interface, run the command "make" in your terminal
    - Absolut is written in C++, and when downloading it, Absolut has not been compiled yet. By running this command, you will compile the source code and produce an executable. This executable is responsible for communication with immuneML.
6. Once compiling is done, you will have an executable called "AbsolutNoLib". You will use the path to this file when filling in a YAML file in part 3.

You have now downloaded Absolut and made it runnable.

Part 2 - download YAML specification file

Before running immuneML, you must create a YAML file to specify an analysis that immuneML will perform. You will be downloading a partially filled-in YAML file based on the documentation. This will already contain the specification for generating a dataset. To download it, follow these two steps:

1. Located in the same link you used to download Absolut, download the file called "specs.yaml"
2. Move the file to a location that you will easily find later

Part 3 - run immuneML with Absolut

You now want to generate a dataset in immuneML and use Absolut to preprocess it before exporting it. To do this, you have to fill in the YAML specification file provided, where your main task will be to define the tool section. This is necessary for immuneML to know what to do when running. To understand how the YAML file is specified, follow the documentation: Link to documentation

1. Open the downloaded specs.yaml file in an editor (such as PyCharm or VScode)
2. The YAML file requires you to fill in the definitions, instructions, and tools sections:
    1. In the "tools"-section: define the tool
        1. When defining a tool, a path to the file used to communicate with immuneML is required. Make sure to add the entire path to the executable located in Absolut that immuneML must communicate with.
        2. Make sure to define the tooltype as "PreprocessorTool"
        3. External tools can require certain parameters to be specified. In this case, Absolut requires you to define parameters under the "params" section. These are as follows:
            - option: repertoire
            - antigen: 1FBI
            - threads: 3
    2. In the “definitions”-section: define the preprocessor
    3. In the “instructions”-section: specify the preprocessing sequence you will be using
3. Save the YAML file. To run immuneML, you will now have to follow these steps:
    1. Through the command line, activate the virtual environment where immuneML is available (the virtual environment that you created for immuneML)
    2. Through the command line, navigate to the folder where you saved the YAML file you downloaded and filled in
    3. To run immuneML, you must run this command: *immune-ml specs.yaml ./analysis_results/*
        - By running the command, you run immuneML with the specified YAML file you created.
        - "./analysis_results/" is a folder immuneML creates at runtime to store the data it produces.

You have successfully generated a dataset in immuneML and preprocessed it with the software Absolut when the final output is “ImmuneML: finished analysis.”

Previous page Next page

Responsible for the form:
jorgeski@uio.no

Nettskjema

The form should be anonymous. **Show more**

immuneML Absolut

80 %

Mandatory fields are marked with a star *

Task questionnaire

Were you able to finish the task? *

◯ Yes

◯ No

Approximately how many minutes did you spend on preparation before starting the task? *

Approximately how many minutes did you spend on doing the task? *

Task description

	Strongly disagree	Disagree	Neither agree nor disagree (neutral)	Agree	Strongly agree
I understood the purpose of the task *	◯	◯	◯	◯	◯
The task description was clear *	◯	◯	◯	◯	◯

Process

This section is about how you experienced using an external tool to preprocess a dataset

	Strongly disagree	Disagree	Neither agree nor disagree (neutral)	Agree	Strongly agree
It was easy to follow the step-by-step guide for the task *	○	○	○	○	○
It was easy to run immuneML with an external tool *	○	○	○	○	○
The process of running immuneML with an external tool was uncomplicated	○	○	○	○	○
I am satisfied with the process of running immuneML with an external tool	○	○	○	○	○

YAML file

This section is about how you experienced filling in the YAML file.

	Strongly disagree	Disagree	Neither agree nor disagree (neutral)	Agree	Strongly agree
I understood what the YAML file was used for *	○	○	○	○	○
I understood the structure of the YAML file *	○	○	○	○	○
I understood where to fill in the necessary fields in the YAML file *	○	○	○	○	○
The YAML file was uncomplicated *	○	○	○	○	○

Was there any parts of the process that was confusing or difficult? *

Was there any parts of working with the YAML specification file that was confusing or difficult? *

Do you have any other comments on how you interpreted using an external tool with immuneML? *

Previous page

Send

Responsible for the form:
jorgeski@uio.no

Appendix F

Interview guide

Interview guide

Goal

This interview aims to get feedback on our artifact from the developers of immuneML.

1. Technical
   a. Consent form
2. Project description
   a. Context
   b. Objectives
3. Process flows
   a. Developers
   b. Users
4. Architecture
   a. General architecture description
   b. Interface architecture
5. Communication
   a. Connection script
   b. Technology
   c. Data types
6. YAML structure
   a. Structure
   b. Naming convention
7. Wrap-up
   a. End the interview

Appendix G

Communication solution - notes

Message libraries	**Benefits**	**Drawbacks**
ZeroMQ	• Universal: API's for "all" programming languages and operating systems. • Simple implementation, supports different messaging patterns and transports. • Backed by a large and active community.	• Dependent on third-party tool.
RabbitMQ	• Provides more robust features for reliable messaging, such as message acknowledgments, persistence, and guaranteed delivery. • Active community with a lot of resources available	• Relatively high complexity - steep learning curve for developers • Can introduce performance overhead compared to more lightweight messaging technologies such as ZeroMQ Can consume a lot of memory and CPU resources • Steep learning curve for developers

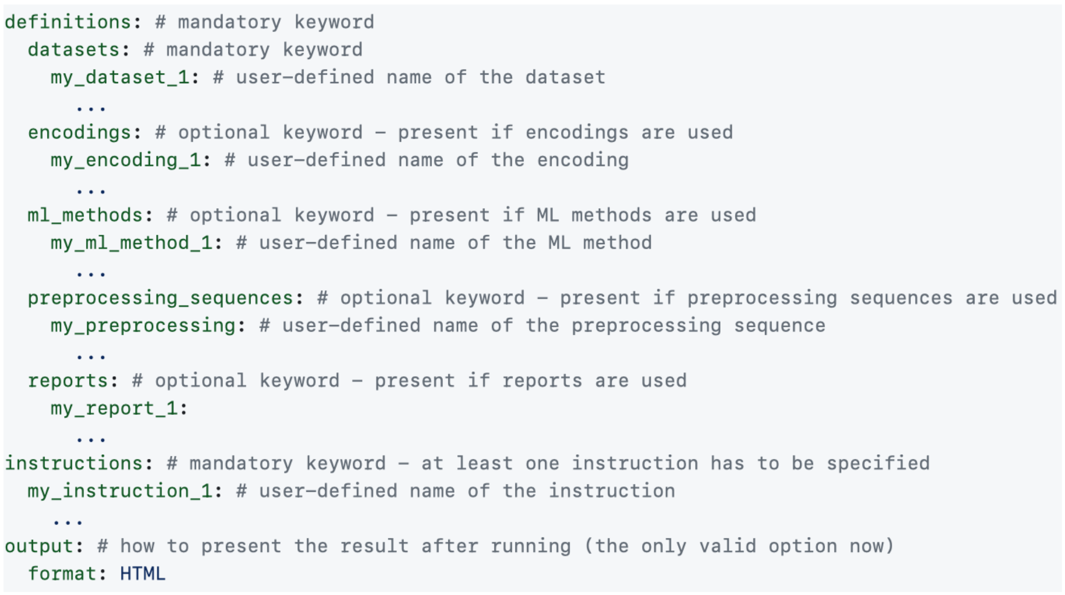
gPRC	• Universal: API's for "all" programming languages and operating systems. Can use protocol buffers (serializing structured data - like JSON but smaller and faster and generates native language bindings. • Cross language compatible. Fast parsing) Uses Interface Definition Language (IDL) to describe the service interface and the structure of the payload.	• Seems more complex to implement and use. • Must define how data has to be structured, Not ideal since we do not know data structure of all functions (e.g. ml method may be different based on method type?)
AMP - Asynchronous Messaging Protocol	• Implementation in a number of programming languages. • Requests and responses are collections of unordered key/value pairs. Keys are strings and pairs can be Standard Data Types(integer, boolean, text, etc.) that is common in programming languages (or other serializations)	• Only Python implementation of tool seems active and widely used.

Apache Thrift	• Universal: API's for "all" programming languages and operating systems. • Similar to gPRC - uses a schema to define interface and functions that automatically creates code in different programming languages. • We are able to provide skeleton code for our interface with correct types and functions. The developer must only add the functionality for each function.	• Seems more complex to implement and use.

Table G.1: Notes from the process of selecting communication solution

Appendix H

YAML file structure



```yaml
definitions: # mandatory keyword
  datasets: # mandatory keyword
    my_dataset_1: # user-defined name of the dataset
      ...
  encodings: # optional keyword - present if encodings are used
    my_encoding_1: # user-defined name of the encoding
      ...
  ml_methods: # optional keyword - present if ML methods are used
    my_ml_method_1: # user-defined name of the ML method
      ...
  preprocessing_sequences: # optional keyword - present if preprocessing sequences are used
    my_preprocessing: # user-defined name of the preprocessing sequence
      ...
  reports: # optional keyword - present if reports are used
    my_report_1:
      ...
instructions: # mandatory keyword - at least one instruction has to be specified
  my_instruction_1: # user-defined name of the instruction
    ...
output: # how to present the result after running (the only valid option now)
  format: HTML
```

Figure H.1: The overall structure of the YAML specification from immuneML documentation [56].

Appendix I

Blueprints

Python

```python
import zmq
import sys

def main():
  port_number = sys.argv[1]

  # Bind to ZeroMQ socket
  context = zmq.Context()
  socket = context.socket(zmq.REP)
  socket.bind("tcp://*:" + port_number)

  # Wait for a message from immuneML
  socket_recv_json()

  # Send an acknowledgment message back.
  socket.send_json("{}")

  program_parameters = socket.recv_json()

  # ---------- Add functionality here ----------


  # --------------------------------------------

  # Send final response
  socket.send_json("{}")

if __name__ == "__main__":
  main()
```

C++

```
#include <iostream>
#include <zmq.hpp>

int main() {

  std::string port_number = argv[1];

  zmq::context_t context{1};
  zmq::socket_t socket{context, zmq::socket_type::rep};
  port_number = "tcp://*:" + port_number;
  socket.bind(port_number);

  // Wait for message from immuneML
  zmq::message_t message_1;
  socket.recv(&message_1);

  // Send an acknowledgement message back
  zmq::message response_1;
  socket.send(response_1);

  // Receive the parameters
  zmq::message message_2;
  socket.recv(&message_2);

  # ---------- Add functionality here ----------


  # ------------------------------------------

  zmq::message_t final_response;
  socket.send(final_response);

  return 0;
}
```

Java

```java
import org.zeromq.SocketType;
import org.zeromq.ZContext;
import org.zeromq.ZMQ;
import org.zeromq.ZMQ.Socket;

public class Main {
    public static void main(String[] args) {
        String portNumber = args[0];

        // Bind to ZeroMQ socket
        ZContext context = new ZContext();
        Socket socket = context.createSocket(SocketType.REP);
        socket.bind("tcp://*:" + portNumber);

        // Wait for a message from immuneML
        socket.recvJson();

        // Send an acknowledgment message back.
        socket.sendJson("{}");

        String programParameters = socket.recvJson();

        // ---------- Add functionality here ----------


        // --------------------------------------------

        // Send final response
        socket.sendJson("{}");

        // Clean up ZeroMQ resources
        socket.close();
        context.close();
    }
}
```