FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

A generic scalable web platform for XAI algorithms

Luís Pedro Viana Ramos

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Mestrado em Engenharia Informática e Computação

Supervisors: Jácome Cunha and Gonçalo Reis Figueira

July 25, 2022

A generic scalable web platform for XAI algorithms

Luís Pedro Viana Ramos

Mestrado em Engenharia Informática e Computação

July 25, 2022

Abstract

Over the years, Artificial Intelligence (AI) has played an ever more significant role in society, from medical applications to video games. It is becoming an essential part of society, whether it is predicting outcomes or making decisions for humans. AI is becoming a central field, and its explainability is often a critical barrier to its wide adoption. The topic of eXplainable AI (XAI) is being explored from different perspectives, and several tools are emerging. However, most of the focus has been on post-hoc explanation models and tools, and explainable-by-design approaches have been overlooked.

This project aims to design and develop the infrastructure and backend of a web platform that allows researchers and algorithm developers to run algorithms to solve predictive and prescriptive problems. The platform allows the parameterization of these algorithms removing the need to change the code whenever a new run is required. It also aims to do this in a scalable way, allowing multiple users to concurrently run their algorithms and search for solutions to their problems, which requires thoughtful planning and implementation of the infrastructure. For this web platform to achieve the flexibility required, a few key factors must be present: efficient use of backend resources when managing the different users' algorithms, a modular design of the XAI algorithms, as well as a modular design of the communication between the different components of the platform. Therefore, domain and requirement analysis is performed to ensure this project meets its goals.

The final platform provides a framework for researchers and practitioners to develop solutions based on symbolic XAI algorithms, whether developing new algorithms or fine-tuning hyperparameters of the models. The platform allows users to create XAI projects, select, customize and train XAI models with a dataset and custom parameterization. Furthermore, it obtains the output of this training process, whether this is a predicted dataset or performance statistics.

The platform is validated by the completion of most of its requirements, as well as for its ability to run different algorithms and the performance of these runs, for which performance tests were conducted. We conclude that there is a cost in both resource usage and time associated with the platform's usage, but it is within acceptable values. The platform was also tested in a live demo with several AI researchers that provided some important feedback for the next steps of the platform.

Keywords: Explainable Artificial Intelligence, Web Platform, Symbolic Learning, Backend, Infrastructure, Scalability

Resumo

Ao longo dos anos a Inteligência Artificial (IA) tem desempenhado um papel cada vez mais importante na sociedade, tornando-se parte essencial da mesma, quer seja a prever resultados, quer seja a tomar decisões pelos humanos. Como tal, a IA está a tornar-se um domínio central e a sua explicabilidade é, normalmente, uma barreira à sua adoção generalizada. O tópico de Inteligência Artificial eXplicável (IAX) está a ser explorado em diversas perspetivas, e várias ferramentas estão a surgir nesta área. Apesar disto, grande parte dos esforços têm sido em modelos e ferramentas de explicabilidade *post-hoc*, e a explicabilidade por conceção tem sido negligenciada.

Este projeto procura desenhar e desenvolver a infraestrutura e o *backend* de uma plataforma *web* que permite a investigadores e programadores de algoritmos correr os mesmo com o intuito de resolver problemas preditivos e prescritivos. A plataforma permite a parameterização de algoritmos, removendo a necessidade de alterar o código, sempre que uma nova execução seja necessária. Estas execuções são feitas de forma escalável, permitindo que vários utilizadores corram os seus algoritmos e pesquisem soluções para os seus problemas concurrentemente, o que requer um cuidado planeamento e implementação da infraestrutura. Para que a plataforma consiga alcançar a flexibilidade desejada, alguns fatores-chave devem estar presentes: o uso eficiente dos recursos, aquando da gestão dos algoritmos dos diferentes utilizadores; um desenho modular dos algoritmos de IAX e da comunicação entre os diferentes componentes desta. Para tal, é realizada uma análise do domínio do problema e dos requisitos necessários para garantir que o projeto alcança os seus objetivos.

A plataforma final providencia uma *framework* onde investigadores e peritos podem desenvolver soluções baseadas em algoritmos de IAX simbólicos, quer estes estejam a desenvolver novos algoritmos ou a aprimorar os *hyperparameters* de modelos já existentes. Esta permite também criar projetos, selecionar, adequar e treinar um modelo de IAX com dados providenciados pelos utilizadores e com parâmetros customizados. A plataforma obtém também os resultados do processo de treino, quer estes sejam uma tabela de dados ou estatisticas de desempenho.

A plataforma é validada através do cumprimento de grande parte dos requisitos, pela sua habilidade de executar diferentes algoritmos, e pelo desempenho destes mesmos, para os quais testes de desempenho foram realizados. Com estes, concluimos que existe um custo tanto a nível de recursos como de tempo de execução associados à utilização da plataforma, mas este encontra-se dentro de valores aceitáveis. A plataforma foi também testada numa demonstração ao vivo com vários investigadores de IA, providenciando *feedback* importante para os seus desenvolvimentos futuros.

Keywords: Inteligência Artificial Explicável, Plataforma Web, Aprendizagem Simbólica, Backend, Infraestrutura, Escalabilidade

*"You should be glad that bridge fell down.
I was planning to build thirteen more to that same design"*

Isambard Kingdom Brunel

Contents

List of Figures

List of Tables

Listings

Abreviaturas e Símbolos

TRUST-AI	Transparent, Reliable and Unbiased Smart Tool for Artificial Intelligence
AI	Artificial Intelligence
XAI	eXplainable Artificial Intelligence
GP	Genetic Programming
API	Application Programming Interface
CLI	Command-Line Interface
SSL	Secure Sockets Layer
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
TLS	Transport Layer Security
OS	Operating System
JWT	Json Web Token
INESC TEC	Institute for Systems and Computer Engineering, Technology and Science

Chapter 1

Introduction

Throughout the years, there have been many developments in Artificial Intelligence (AI) [17]. These developments have come to introduce AI into our everyday lives. However, many of these algorithms leave much to be desired in terms of explainability of the outcome [25]. For this reason, a new research field has emerged which focuses on this problem: eXplainable Artificial Intelligence (XAI). This new field came with a new set of problems, mainly a need to specify how the explainability of these algorithms work [5, 23]. These explanations are often achieved by providing the user with certain parameters that can be specified and help the algorithm obtain an explainable solution. Along with this, the fact that the algorithm must give an output and an explanation for it means that these XAI algorithms require more resources to run.

As such, with a need for ever more explainable solutions, companies and researchers alike are using more and more XAI algorithms [10, 25]. However, these algorithms use different programming languages and frameworks and have different requirements [23], making the process of using a new algorithm to solve a specific problem very time-consuming. Furthermore, with new algorithms appearing frequently, the analysis and comparison of new solutions only add more and more time to an already complicated project.

Another aspect that is often disregarded, but is very present in AI-based solutions, is the training time of AI models. This time might be impacted by the complexity and extra processing power required by XAI algorithms, meaning that a single training session might take more than a week to complete [4].

1.1 The TRUST-AI Project

To tackle the challenges posed by the development process of XAI algorithms, a new platform called Transparent Reliable and Unbiased Smart Tool for Artificial Intelligence (TRUST-AI) is being designed and developed. This tool will allow researchers to efficiently develop XAI algorithms and solutions in a standardized way by providing a common platform on which to test new algorithms, optimize their hyper-parameters and receive insights on the models output by the algorithms.

TRUST-AI is a part of the European project with the same name [45], which is funded by the European Union's Horizon 2020 research and innovation program. This project, led by the Institute for Systems and Computer Engineering, Technology and Science (INESC TEC), aims to "bridge the gap between the analytical expressions derived from theory and the numerical models obtained with Machine Learning. A novel paradigm will be developed whereby humans and machines can collaborate and discover new solutions" [45]. In order to do this, the project is split into the following main goals:

- Use explainable AI algorithms to solve problems to obtain explainable solutions that can be used with traceability only provided by XAI algorithms.
- Provide deep and meaningful explanations for all solutions provided by the algorithms.
- Allow users to delve deep into the proposed solution and customize it while always knowing its effect on the final solution.

The TRUST-AI project has three different use cases: discovery and treatment of paraganglioma cancer (healthcare sector), pricing of grocery deliveries (online retail sector), and prediction of energy consumption (energy sector).

With a focus on explainable AI, the TRUST-AI platform is divided into four different components, as seen in Figure 1.1, which are described as follows:

- **AI Engine** - Component in which the different algorithms of the platform run.
- **Cognitive Models** - Provide explanations and insights of received models.
- **Interfaces** - Provide an interface for the management of algorithms, models and their explanations.
- **Application Server** - Responsible for the communication between all other components, as well as, the storage and management of data from each component.

1.2 Research Objectives

With the division of the platform into the four different components, this dissertation focuses not only on the Application Server component but also on the management of the AI Engine and, with this, the infrastructure that supports all other components.

TRUST-AI aims to provide a platform that gives users a standardized environment to run their XAI algorithms and research solutions to their problems. As such, the main objectives of the thesis are:

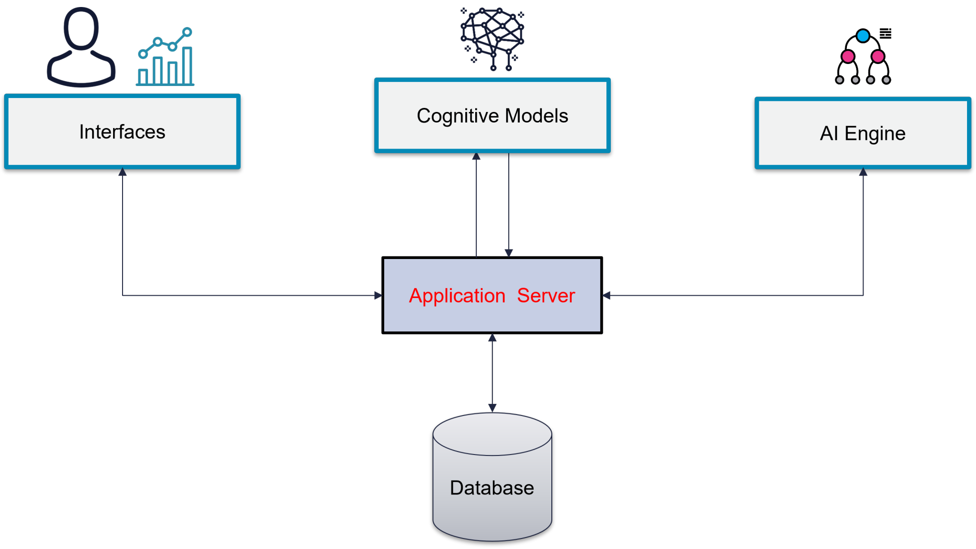

Figure 1.1: Overview of TRUST-AI platform components.

- **Design a scalable infrastructure** - XAI algorithms require more resources than the usual AI algorithms. For this reason, it is essential to have a scalable infrastructure capable of handling multiple algorithms simultaneously while taking full advantage of the available resources.
- **Control the XAI algorithms in the platform** - The TRUST-AI platform should control the algorithms' execution and acquisition since these algorithms can either come from the user or be specific to the platform.
- **Abstract the different paradigms of AI** - TRUST-AI should handle both predictive algorithms, where the objective is to predict a certain value or values and prescriptive algorithms, whose objective is to optimize a given function.
- **Manage communication between components** - Since the backend communicates with all other components, as seen in Figure 1.1, it is crucial to develop a communication base capable of evolving and changing with the different algorithms and cognitive models used.

The platform focuses on abstracting the different paradigms of AI, their algorithms, and frameworks so that the users can create and run their algorithms without changing the TRUST-AI tool regardless of their implementation. As such, allowing the users to compare models for an optimized solution since a model is only one of the outputs of the algorithm, which can be optimized by changing the algorithm's parameters or by hand. TRUST-AI also focuses on scalability, allowing different users to run their models concurrently, utilizing the available resources in an optimized way.

This dissertation's work is focused on designing the infrastructure that houses the different components of TRUST-AI, developing the Application Server component, and managing the scalability of the AI Engine by controlling the entire AI Engine component, including the introduction of the user algorithms.

The implementation of this tool also allows the study of current scalability methods and technologies, as well as methods of abstracting and generalizing the different paradigms of AI. It also analyses and uses technologies that allow for a dynamic introduction of new algorithms without recompiling and deploying the platform.

The development of this work follows an agile methodology, dividing the development period into sprints that categorize and mark each step in the development of the TRUST-AI tool. The Interfaces component for this tool was designed alongside the infrastructure and backend design in a separate dissertation project entitled *Interfaces for human-guided AI* [22]. As such, this work focuses on the backend and infrastructure while developing the communication with the frontend, but it does not tackle the platform's user interfaces.

1.3 Structure

This dissertation is divided into the following chapters:

Chapter 1, "Introduction" - A definition of the problem, why it is relevant and how we intend to solve it.

Chapter 2, "State of the Art" - A review of literature regarding AI and XAI solutions, as well as implemented tools and frameworks with similar purposes to the TRUST-AI platform.

Chapter 3, "Specification" - Detailed discussion of the problem along with the proposed solution given in the form of requirements imposed on the platform, types of users and its use cases.

Chapter 4, "Technologies" - A review of possible technologies that can be used in the platform development.

Chapter 5, "Platform Implementation" - Detailed discussion of the implementation process and presentation of the developed platform functionalities.

Chapter 6, "Platform Evaluation" - Presentation of the resulting platform and of aspects that were achieved.

Chapter 7, "Conclusions" - Conclusion of work done, with brief discussion of major points and of future work.

Chapter 2

State of the art

This chapter discusses key concepts that must be understood before developing a platform for XAI algorithms. We start by analyzing the XAI domain, reviewing the literature on the current XAI algorithms in Section 2.1 and their development process in Section 2.2. In Section 2.3 we discuss some of the tools and frameworks already developed to solve similar problems, including their similarities and differences to the TRUST-AI platform. Finally, in Section 2.4 we analyze existing types of scalability and how their concepts can be applied to the platform.

2.1 Explainable Artificial Intelligence

Before starting to analyze and discuss existing frameworks, a study of how XAI algorithms work, what they require to run, what they output, and how they are developed. More specifically, we studied algorithms that generate numerical models and expressions, such as Genetic Programming (GP) algorithms, as these are the ones for which the TRUST-AI platform is mainly developed. This study allows us to better understand the researcher's point of view when introducing new algorithms, so we can better design the platform to accommodate the standard algorithm design principles.

As the name suggests, the key requirement of an XAI solution is explainability. "Explainability is a means to enhance user trust in the models" [2]. As such, there are two possible approaches to XAI as referred by [5]: transparent machine learning models; and post-hoc explainability techniques.

2.1.1 Transparent Machine Learning Models

"A model is considered to be transparent if by itself it is understandable" [5]. The explainability is typically obtained via a rule set or mathematical formula that the models generate to obtain the solution or a small set of values that are always used in the same manner and, as such, can be easily calculated and understood by a human. Many algorithms fall within this category, as are the cases of:

- **Genetic Programming** (GP) is an evolutionary computing technique developed using the principles of genetics and natural selection [19]. Each model output by this algorithm is represented as combinations of user-defined operators and terminals [13]. Which are then combined using mutation processes repeated for a certain number of user-defined generations and individuals in the population. The final model is a series of operators and terminals, which can be displayed in either a formula or code format, that can be studied by humans.
- **Decision Trees** are sequential models, that combine a sequence of logical tests on the available variables [14, 20], such as equals and less then operations. These models are classified as explainable since logical tests can be easily understood by humans.
- **K-Nearest Neighbour** algorithm is an effective algorithm that does not make presumptions on the provided dataset [21]. Instead, it categorizes the data points into classes using a labeled training dataset, which is the basis of its explainability.

This type of explainability can be expected in the TRUST-AI framework. However, it should come from machine learning models like the ones described above. It is not an inherent feature of the TRUST-AI platform. Furthermore, it is not a concern of the platform but of the algorithm used to obtain a solution.

2.1.2 Post-hoc Explainability Techniques

Post-hoc explainability techniques resolve around obtaining a rule set or importance value for each feature [24] that is used in the machine learning model. These techniques add explainability to an already existing solution that might not be explainable by itself. Some of the most known techniques, that mentioned in [16, 12], are: feature importance based explanations; rule-based explanations; saliency maps; counterfactual explanations; summaries of counterfactuals.

The TRUST-AI framework plans to deliver explainability through post-hoc explainability techniques, having a specific component already planned for this purpose, the Cognitive Models component illustrated in Figure 1.1. However, the development of this component is beyond the scope of this thesis.

2.2 AI Algorithm Development Process

There have been many studies into the development of AI projects, these have created methodologies such as CRISP-DM, SEMMA and KDD [3]. However, all these different methodologies have some common steps.

In *An Artificial Intelligence Life Cycle: From Conception to Production*, De Silva and Alahakoon [8] describe the AI development process in 17 different steps. These start from the inception of the problem all the way to the usage of the model obtained from the machine learning algorithm. However, since the TRUST-AI platform is designed for the researcher and, as described in the paper, the AI/ML Scientist, we will only be looking at the processes that occur or affect the development of the algorithm:

- **Data augmentation** - In this step the researcher's goal is to enhance the data in a way that will produce better results. This phase is crucial and can be done in many different ways, whether it is done just before the data is used by the algorithms or, in a more static approach, to the initial data file. The difference being that in the first method the data is only changed temporarily and in the second it is done permanently. In the TRUST-AI platform this step should be done by using the static approach, while having the possibility of reverting changes. This allows users to experiment and change the dataset according to the results obtained.
- **Build initial AI model** and **develop benchmark** - These steps were put together because in the TRUST-AI platform these are the same step, since building and launching a model produces a benchmark that is always available and comparable. Here the objective of the researcher is to run a model that already solves the problem in a simple but effective way and use it to compare with future models.
- **Build multiple AI models** - Having a benchmark, the next step is to create more complex AI models that better suit the problem. Focusing on features that the other models failed to capture. This step is iterative and, as such, should be supported by TRUST-AI in an iterative way, by allowing training sessions to be started from a previously configured session.
- **Evaluate primary metrics** - When building and using an AI model the first steps taken to evaluate it are usually error and accuracy/recall measures. These allow the researcher to know how successful the model was at predicting a certain known outcome. For this to be able to be done, the data must be divided into two parts: train data and test data. With this, the resulting model from the training session with the train data is tested against the test data and the measures are obtained. This step is crucial for the researcher as it is here that the performance of the model is tested. For the platform, this step must be done by processing some of the metrics that the algorithm outputs, along with obtaining the logs of the algorithm run, which might contain this information.
- **AI Model Explainability** - In this step the main focus is on the explainability of the obtained solution. In terms of rule sets and mathematical expressions it is important to note that the magnitude of the solution might get too big and not be explainable anymore. In this step the model might also be analysed using a post-hoc technique. As such, TRUST-AI should allow for the researcher to either visualize the mathematical solution or rule set and apply post-hoc techniques with the cognitive models component.

- **Evaluate secondary metrics** - Having validated the model and its explainability the next step is to evaluate another set of metrics that accompany any software: CPU usage and memory usage. With this, the researcher is able to understand how efficient the utilized algorithm is. In TRUST-AI these metrics should be automatically calculated since the resources used by an algorithm are an important factor to the scalability of the platform.

2.3 Existing Frameworks and Tools

Throughout the years, some tools have been developed and brought to the public for researching and obtaining solutions via the utilization of XAI algorithms. There have been many approaches to developing these tools. Some focus more on the model [43, 31, 34], while others focus on providing a complete development tool for both the model and the algorithm [28, 35, 38].

Many of these tools rely on simply providing a standard algorithm that can solve most problems. We will be looking at a few examples of this approach that are more closely related to our work, such as TuringBot [43] and Eureqa [31], which is no longer commercially available. Both of these examples allow for the specification of the problem, but in a limited way, as these problems can only be about predicting values, limiting the types of problems that can be solved. It is also important to note that these tools are not research tools since they focus on the model instead of on how the model is obtained via the algorithm. This is an important factor since it is something that TRUST-AI will focus on, giving the researcher complete control over the algorithm.

Another approach is to fully design and develop modular algorithms from the start and integrate them with the interface. This approach was the one used in the development of the HeuristicLab [35] framework and tool. Despite the existence of other tools [26, 28, 38], HeuristicLab is the one that presents most of the features that we want and that deals with the same problems as TRUST-AI. HeuristicLab allows the user to customize every aspect of the algorithms used and the problems they are trying to solve. This tool takes a step further and allows the user to fully implement an algorithm that can be used to solve the problems, which gives the researcher the ability to try different algorithms and approaches to the same problem. However, it provides this functionality by allowing the user to program the algorithm in C#, which can be seen as a significant downside since it requires the researcher to be familiar with the language and paradigm it works in. With all these functionalities, the HeuristicLab framework becomes quite complex and challenging to use for anyone new to the framework.

Even though Eureqa, TuringBot and HeuristicLab provide some if not most of the functionalities we are looking to implement with the TRUST-AI platform, one key aspect that the TRUST-AI tool is looking to provide is that it is an online tool. Its algorithms do not run on the user side, keeping the user's computer fully operational during training and testing sessions. Since a single run of an algorithm can take weeks to finish, this is an essential requirement for researchers dependent on their personal computers.

2.3.1 General Purpose Platform Solutions

One possible solution for researchers is the use of general purpose platforms to run their algorithms, for instance, Google Cloud[1], AWS[2] or Azure[3] could be used. These platforms can run any application, and also provide additional services that might complement it, for instance, storage services. Despite this, it can be very difficult to setup a single algorithm in their infrastructure, manage it and obtain its outputs. For this reason, general platform tools are not very used by researchers due to having to learn how to take advantage of the platforms. The TRUST-AI platform distinguishes itself from these by providing a way of running XAI algorithms and obtain their outputs in a standard and easily implemented way, removing the overhead of having to learn every aspect of a general purpose platform. Another important aspect is that TRUST-AI provides additional benefits besides just running the algorithms. It allows users to visualize algorithm outputs and receive insights and explanations on the models, which would have to be done by the researcher on their own if they chose to use a general purpose platform to run their algorithm.

2.3.2 Gap Analysis

To better understand the platforms' and tools' strengths and weaknesses, we present Table 2.1. In it, we can see that none of the platforms match TRUST-AI's features. This is because the TRUST-AI platform is being developed to solve the problem of having multiple algorithms implemented in multiple languages while also visualizing specific types of results that algorithms output. Along with this, TRUST-AI also aims to be an online platform, requiring no resources from the user, which in a research context, allows researchers to continue working even when running algorithms.

2.4 Scalability

With an ever increasing user base in most technologies around the world, there have been many developments in the scalability of products and the infrastructure itself. To better understand the scalability concepts they were abstracted to a cube called the scale cube, first introduced in the book *The Art of Scalability* [1]. In this cube the types of scalability are defined in the three different axes:

- X Axis (Section 2.4.1) - Replication of instances/services onto which the work will be divided.
- Y Axis (Section 2.4.2) - Division of tasks into services by knowing what type of operation and resources are needed, dividing work into steps to be done at each different service.
- Z Axis (Section 2.4.3) - Division of data across the available services to do parallel work.

[1]Found at `https://cloud.google.com/`
[2]Found at `https://aws.amazon.com/`
[3]Found at `https://azure.microsoft.com/`

Table 2.1: Features of tools and platforms including TRUST-AI.

Name	**Algorithm**			**Processed Data**	**Resource Usage**	**Explanations**
	Adding	**Coding Language**	**Type**			
Turing Bot	No	Python	Predictive	Yes	Personal	Yes
Eureqa	No	#N/A	Predictive	Yes	Personal	Yes
H20	No	#N/A	Predictive	Yes	Server	Yes
Heuristic Lab	Yes	C#	Any	Algorithm Defined	Personal	Yes
Google Cloud	Yes	Any	Any	No	Server	No
AWS	Yes	Any	Any	No	Server	No
Azure	Yes	Any	Any	No	Server	No
Cloud AutoML	No	#N/A	Predictive	Yes	Server	Yes
KubeFlow	Yes	Any	Any	No	Server	No
TRUST-AI	Yes	Any	Any	Framework	Server	Yes

2.4.1 The X Axis

The X axis scalability is achieved by having multiple clones of a service and having a load balancer that controls to which service goes what request. This type of scalability is easily achieved and implemented with current state of the art technologies, and is called elastic scalability in cloud platforms as it will increase the number of resources available to the application according to their usage.

Some of the main cloud platforms that offer elastic scalability are AWS Elastic[4] service and Google Cloud through the commonly utilized tool called Kubernetes[5].

This type of scalability is easy to implement with the right technologies, since it only implicates that the existing services are cloned. It allows for transactions to scale because it sends each transaction to one of the services running. However, this type of approach is often costly because it relies on adding more services whenever it is required.

This scalability type is the one we are looking for when managing the different algorithms that will be available in the TRUST-AI platform, as such, Kubernetes will be used along with a mechanism to control how elastic the algorithm management can be.

2.4.2 The Y Axis

The Y axis scalability is identified by the services in which an application is divided, whether it is just one that controls the whole application, or many different services that control different aspects of the application. This type of scalability is harder to achieve than the X axis type simply because it requires that the functionality of the application is divided into different steps to be executed by

[4]Found at `https://aws.amazon.com/elasticbeanstalk/`

[5]Found at `https://kubernetes.io/`

different services. As such, Y axis scalability requires that the system design is modular enough to be split into different services, thus putting a lot of emphasis on the design of the system itself. It also might take additional time to implement when compared to the other types since it requires that parts of the system become isolated and have their own defined resources, something that can be very difficulty for complicated functionalities.

This type of scalability is an inherent property of the Microservice Architecture [40], since this architecture focuses on dividing the application as a whole in different services that provide different functionalities.

2.4.3 The Z Axis

The Z axis scalability is represented by dividing the data received so that each service clone only handles part of the data. This is the same concept as parallel optimizations where the data processing is divided according to the number of CPUs present in the system. However, since we are talking about servers and server clouds there is likely more benefit from taking advantage of multiple servers instead of a single one. This scalability type has some benefits which are: improved cache utilization and reduced memory usage and I/O traffic; improved transaction scalability; and reduced utilization of each server per request. Despite this, the drawbacks to this type of scalability outweigh the benefits since to implement Z axis scalability the application's complexity increases a lot, due to having to create a partitioning scheme for the data. This type of scalability will likely not be used in TRUST, however it is still important to allow algorithms to scale by allowing parallel optimizations, taking full advantage of the resources available to the server in which they are running.

Chapter 3

Specification

The TRUST-AI platform is being developed as an assisting tool to study and solve problems that require not only a solution but also an explanation. For this reason, the platform is being developed with a heavier focus on XAI algorithms. These algorithms, much like AI algorithms, can be divided into four different steps, which help us understand the underlying processes that an algorithm run is comprised of:

1. Data acquisition - this step usually involves a dataset that has been studied and prepared for the algorithm execution. Which is then passed to the algorithm.
2. Algorithm configuration - the configuration of the algorithm is set to optimally run the algorithm according to the dataset, the problem that is being studied and the resources available in the system.
3. Algorithm execution - the algorithm code is executed using the dataset and configuration provided, producing results which are placed in a known location.
4. Results gathering and study - the algorithm execution results are obtained and processed, first by the algorithm and then by the user to more easily compare the solutions found.

In order to incorporate these steps into the TRUST-AI platform, we start by defining the platform's requirements in Section 3.1. Next, we identify the different actors that perform these steps in Section 3.2, and distill both the requirements and actors into the different use cases of the TRUST-AI platform through user stories in Section 3.3. Finally, we present a simplified workflow of the TRUST-AI platform in Section 3.4 to better understand the functionalities required.

3.1 Requirements

The TRUST-AI platform must also comply with specific requirements to fulfill the intended goals. These requirements come first in the form of functional requirements that the application should support and then are distilled down to system requirements that should be taken into consideration when designing the platform. The functional requirements can be seen in the following table 3.1.

Table 3.1: Functional requirements of the TRUST-AI platform.

RID	**Name**	**Description**
R1	Add algorithm	Add a new algorithm to the platform, whether before the platform is launched or while it is running.
R2	Define possible parameters of the algorithm	Define the possible parameters that the algorithm can receive to avoid errors and wrong parameters
R3	Parameterize algorithm	Parameterize an algorithm according to the specification provided by it
R4	Upload files	Upload files that the algorithm requires to successfully run (usually a dataset)
R5	Run algorithm	Run an algorithm with the files and parameters chosen
R6	Save algorithm data	Save the data generated by an algorithm run, includes logs and files
R7	Process algorithm data	Process known algorithm result files into data structures that can be used in the frontend and in the Cognitive Models
R8	Track algorithm run	Keep track of the status of the algorithm run and any intermediary results
R9	Save specific results	Save specific models that were produced by the models or that the users defined themselves
R10	Run model in algorithm	Run a specific model in an algorithm to obtain its metrics
R11	Model explanations	Provide model explanations by interfacing with the Cognitive Models

Despite the amount of functional requirements, the TRUST-AI platform must ensure a quality service to be useful for researchers. Therefore, the system must be designed with the following non-functional requirements:

- **Performance** - One important distinction that TRUST is trying to bring to researchers is that they do not need to run their algorithms in their own computers. However, this functionality is only useful if the algorithm runs do not have an extreme overhead in their execution times. As such, the platform should optimize its resource usage while running the algorithms as fast as possible.
- **Scalability** - Even though performance is an important requirement for the TRUST-AI platform, it is important to have this performance not only for one execution of a single algorithm, but for multiple algorithms. For this to happen and to not have to wait for other users' algorithms, TRUST should implement a scalable infrastructure were algorithms can run in separate instances.
- **Usability** - Introducing algorithms to the TRUST-AI platform should be a simple process. The development of XAI algorithms for personal use and for use in the platform should

be as similar as possible. Without this requirement the whole purpose of the TRUST-AI platform is defeated as its main mission is to provide a web platform to run researchers' algorithms. In order to comply with this requirement a well designed framework should be built so that the environment created by the platform is as close as possible to standard.

- **Security** - As this platform is to be used by researchers with sensitive data, it is important to design a platform that handles each user's data in a secure way. This requirement is especially important for use cases such as the health-care sector as the data being introduced into the platform is extremely sensitive and if leaked could have severe consequences.
- **Maintainability** - With different components being developed by different partners and at different times our infrastructure and application server component will need to be maintained with the evolution of the platform. For this reason it is important to develop a modular infrastructure with as little dependencies as possible to specific components. It is also important to develop the Application Server with a degree of abstraction big enough to accommodate changes in other components without big changes being required in the code.

3.2 Actors

From the requirements and the platform specification the different actors that will perform actions in the platform can be specified as:

- The **Algorithm Developer** is the one responsible for introducing algorithms into the platform. This actor has the most knowledge of AI methods and techniques.
- The **Model Developer** understands the problem and the algorithms that are present in the platform, they customize and configure the algorithms to the problem and produce results. This actor will likely be the one that interacts the most with the platform.
- The **Domain Expert** has expertise in the domain of the problem that is being solved. As such, he will interface with the results of the algorithms and their explanations while comparing them with current practises and ideas.

These actors interact in different stages of a project, as can be seen in Figure 3.1.

3.3 User Stories

Having defined the platform's requirements, the platform's use cases were detailed in the form of user stories. A user story is a brief description of a software feature detailed from the user's perspective. Each of these user stories is related to at least one requirement, making it easier to track the completion of each platform requirement. Table 3.3 defines the TRUST-AI platform's user stories and the requirements that they relate to.

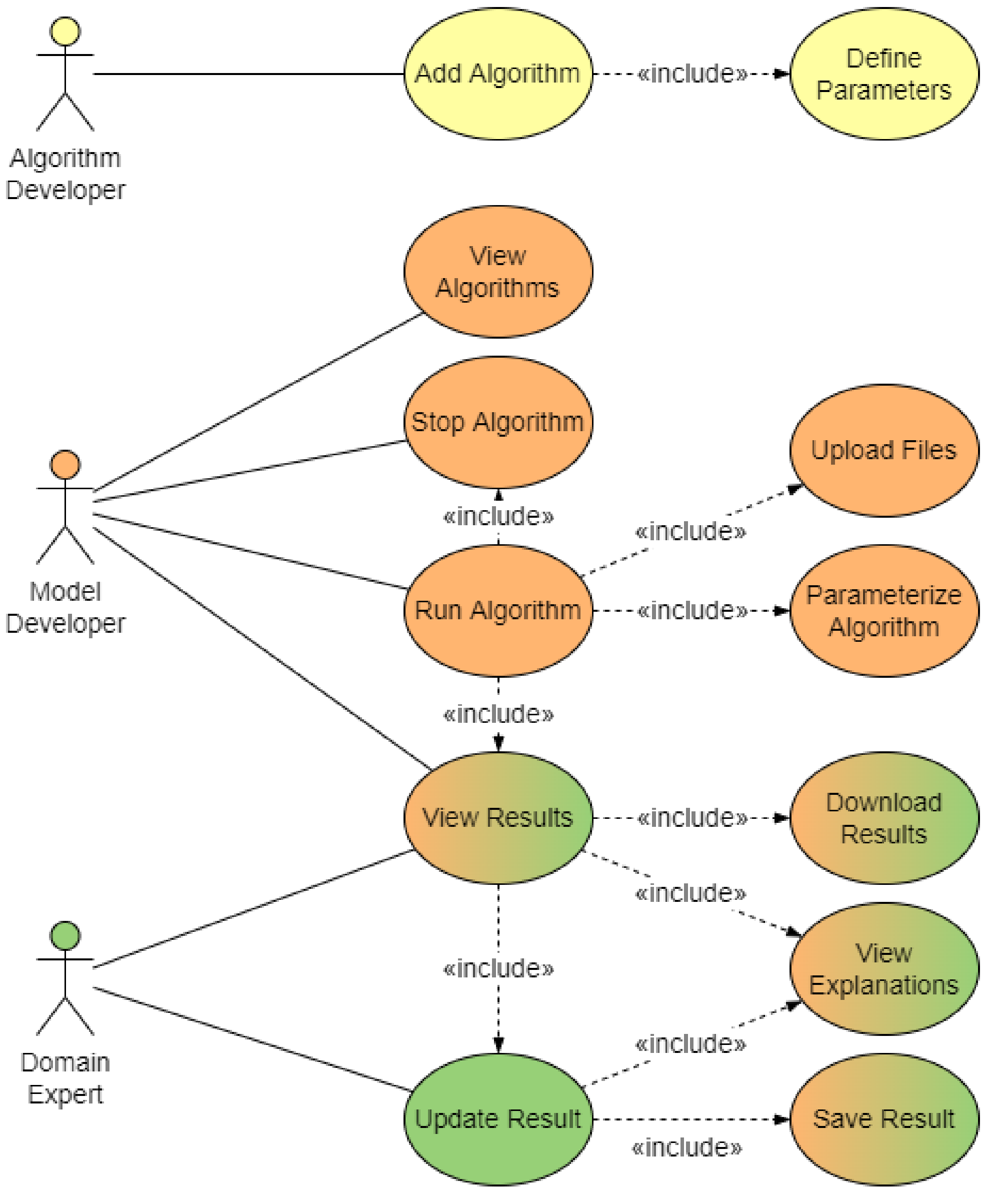

Figure 3.1: Use cases of each different actor and their relationships.

Table 3.2: TRUST-AI platform user stories.

UID	RIDs	Done	As a	I want to	So that
US1	R1	Yes	Algorithm Developer	Add an algorithm to a running platform	I can use it to solve a problem
US2	R1	Yes	Algorithm Developer	Add an algorithm to the platform's files	When I launch the platform the algorithm is already compiled

Continued on next page

Table 3.2 – continued from previous page

UID	RIDs	Done	As a	I want to	So that
US3	R1	Yes	Algorithm Developer	See if the algorithm was added successfully	I can correct any errors
US4	R2	Yes	Algorithm Developer	Define the parameters that my algorithm may receive	Users can customize it to their problems
US5	R5	Yes	Model Developer	See the algorithms that are available in the platform	I can choose one that fits my problem
US6	R4	Yes	Model Developer	Upload a file	It can be used in an algorithm run
US7	R3	Yes	Model Developer	Set the parameters that the algorithm will receive	I can customize the algorithm to my problem
US8	R5	Yes	Model Developer	Run an algorithm with a file and parameters of my choosing	I can obtain results specific to my problem
US9	R5	Yes	Model Developer	Stop an algorithm run	I can terminate a wrongly configured run
US10	R8	Yes	Model Developer	See the status of my algorithm runs	I can keep track of their progress
US11	R8, R6	Yes	Model Developer	See the logs of my algorithm runs	I can keep track of their progress
US12	R6	Yes	Model Developer	See and download the results that the algorithm is outputting while it is still running	I can keep track of how the run is going
US13	R6	Yes	Model Developer	See and download the final results of my algorithm run	I can process them in my own way
US14	R7	Yes	Domain Expert	Get the processed results from an algorithm run	I can compare them with other runs
US15	R7, R9	Yes	Domain Expert	Save processed results that I find of value	I can compare them with other algorithm runs
US16	R11	No	Domain Expert	Receive explanations about the results of an algorithm run	I can better understand the results
US17	R10	No	Domain Expert	Change a processed result and test it in the algorithm	I can validate the changes

3.4 Workflow

The expected workflow of the TRUST-AI platform is very much modeled after the workflow of the development of an XAI project, as seen in Figure 3.2. The user starts by creating a User account, giving the user access to the platform and the information regarding the algorithms. He can then create a project where his different algorithms runs will be stored. Here he can also upload files that the algorithms need, such as a dataset. After that, he can create a session by choosing one of the algorithms and a file to send to the algorithm. In this session, the algorithm run data will be stored and visible. After setting the algorithm's parameters, he can begin the algorithm run, whose progress can be tracked using the session's state, logs, and output files. During the algorithm run and after it is finished, the user can view and save the algorithm's results, allowing him to compare the results from different sessions.

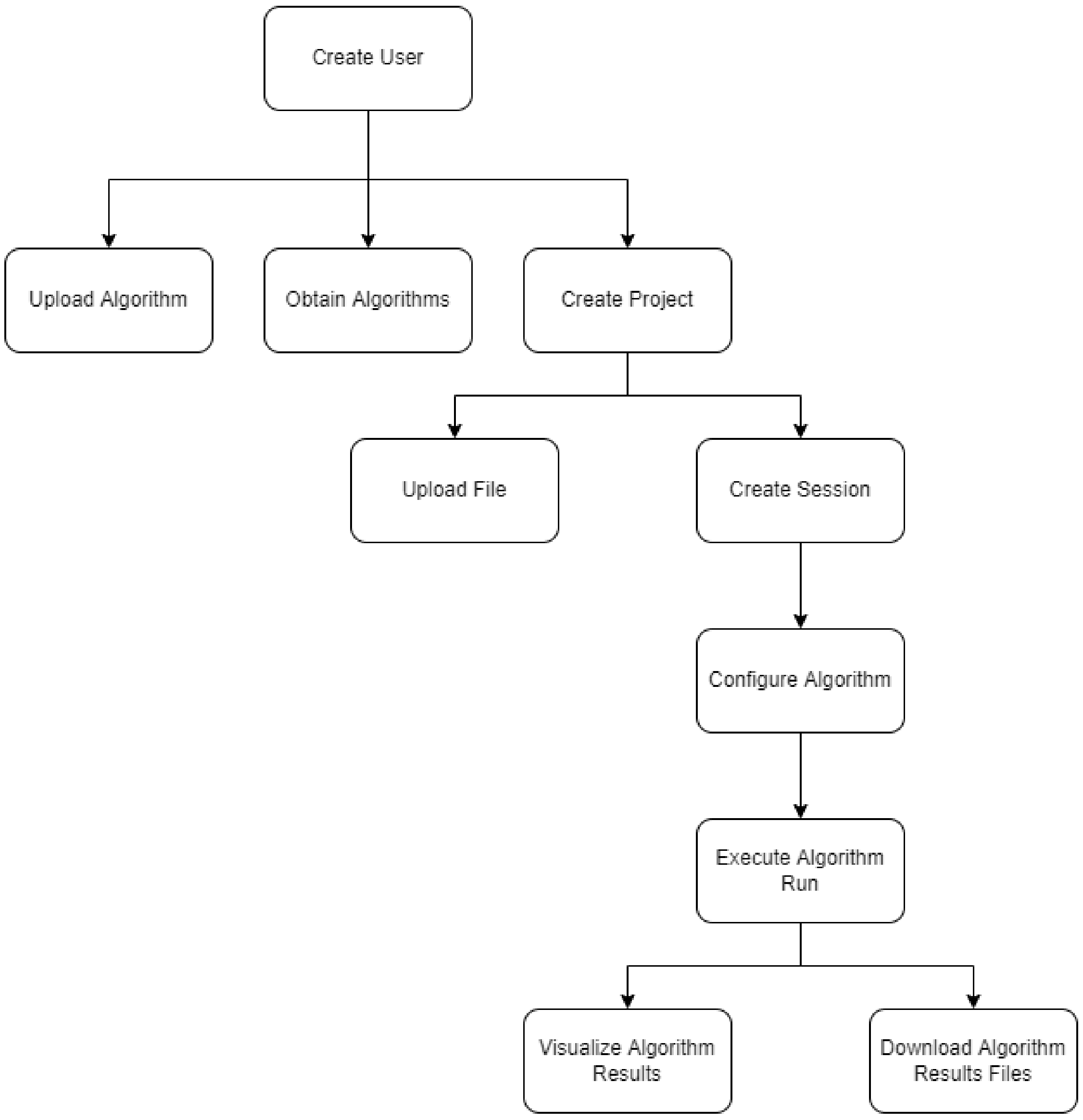

Figure 3.2: Simplified workflow of the TRUST-AI platform.

Chapter 4

Technologies

The TRUST-AI platform requires plenty of technologies to incorporate the different components in a maintainable way and implement the functionalities needed. For this reason, in this chapter, we review and discuss technologies that allow us to abstract the different components of the TRUST-AI platform in Section 4.1, along with server management technologies in Section 4.2, that can be used to configure the environment and infrastructure required by the platform. We also discuss the technologies that are used for the abstraction of the different algorithms that the platform uses in Section 4.3. Finally, the technologies used to implement the backend of the TRUST-AI platform are discussed in Section 4.4, as well as the database that we chose to store the platform's data in Section 4.5.

4.1 Component Abstraction

In the development of TRUST-AI, we try to abstract as much as possible the different components so that the tool becomes highly adaptable to any algorithm and utilization. To do this, we develop each component separately to reduce the dependencies on other specific services, which is done by relying on the functionalities available in the platform as a whole. For this, we use two tools with different purposes.

The first tool used is Kubernetes[1], "an open-source system capable of automating deployment, scaling, and management of containerized applications" [39]. This tool is being used with Docker[2], an application that uses OS-level virtualization to deliver software in containers [30]. With these technologies, each service has its environment and is free to use its allocated resources since Kubernetes provides the developer with some tools to manage each service. It also provides "out of the box" scalability with replicas that can be set for specific services, making it a very useful tool in terms of scalability. It also automatically handles inter-service communication through

[1]Found at `https://kubernetes.io/`
[2]Found at `https://www.docker.com/`

the HyperText Transfer Protocol (HTTP). Kubernetes also provides a development environment that mimics the production environment, making it easier to verify and test before sending it to production. Along with this technology, we are also using *NGINX Ingress*[3] which provides a security layer through the use of the Transport Layer Security (TLS) protocol, and serves as a load balancer, distributing the different requests to the respective services and their replicas. This tool has direct integration with Kubernetes allowing for faster development of the TRUST-AI platform.

The second tool is NATS[4], as it is important to have a communication tool that standardizes most if not all communications between the TRUST-AI components, abstracting the different services and their functionalities. NATS provides communication between services as events that trigger certain functions, which allows each service to define the intended functionality independently. It is possible to abstract the different services by communicating with components that provide the required functionality and without knowing what specific service we need to communicate with. This tool is developed for distributed systems, precisely the system proposed with TRUST-AI. This tool requires an additional service to be run, more specifically NATS JetStream[5], which is the NATS persistence engine that provides streaming, message, and worker queues with At-Least-Once semantics [37].

4.2 Server Management

In order to configure servers and manage their state, a management tool is needed. In addition, we are looking for a tool that is not only configurable for one single deployment but flexible enough to be used easily by others when deploying TRUST-AI in their own environment/servers. As such, the following tools are analyzed, as they provide infrastructure as code:

- **Terraform**[6] allows developers to write infrastructure as code for cloud deployments using declarative configuration files. It does this through a specific language: HashiCorp Configuration Language. This means that in order to start developing the infrastructure code we need to learn a new language which can take its time and comes with its own set of problems. This tool divides the work into 3 different steps: write, where the developer writes the declarative configuration file; plan, which allows the developer to view the execution plan before changing the infrastructure; and apply the configuration to the servers. With this, it is clear that some problems might arise with an agile methodology as the tool does not handle a continuously changing structure very well. However, this tool provides a free to use version which allows the management of self-own servers which is what we are looking for.

[3]Found at `https://kubernetes.github.io/ingress-nginx/`
[4]Found at `https://nats.io/`
[5]Found at `https://docs.nats.io/nats-concepts/jetstream`
[6]Found at `https://www.terraform.io/`

- **Pulumi**[7] is a similar tool to Terraform as it also is developed for cloud deployments. This tool allows us to write the infrastructure in one of the following commonly used programming languages: TypeScript in Node.Js; Javascript; Python; Go; or C#. For this reason this tool can be very easy to pick up and start using if the developer already knows these languages, without having too much downtime learning the syntax. This tool also provides a self-hosted service, which is desired since the health care use case requires that TRUST-AI is deployed and used solely on private servers of the health care institute due to data security reasons. Having said this, Pulumi provides this service as a premium, meaning it requires a constant upkeep in order to be used.

- **Ansible**[8] provides infrastructure management through declarative files, the same way Terraform does. It does this while knowing the full state of the current deployment, meaning the developer only writes the intended state and Ansible will automatically change the required servers to comply. This tool manages all kinds of infrastructures, from cloud servers to containers, meaning it can manage a private server as well. With this tool the infrastructure is declared using YAML[9] files which can be seen as an advantage, as we are using Kubernetes which also uses YAML files to declare the services in use, making it easier to use once we know the syntax used.

These technologies allow server configuration, but in terms of simplicity for the development period, we will be using Skaffold[10] which provides an easy setup of Kubernetes for development. This tool allows us to describe what services we want to initialize and how to do it via a YAML file, which is important since there is a need to persist data within the server. At the same time, the Skaffold configuration can be made with a similar setup with the same results, but without having to worry about many of the problems that come with a production server.

4.3 Algorithm Abstraction

The abstraction of the different algorithms will be done using Docker images. Docker allows code to be portable from one machine to another without worrying about differences in operating systems or even packages installed in the machines. An algorithm can be inserted into the platform via a Docker image. However, an algorithm requires more than just running its code. For this reason, in order to have multiple different algorithms with different configurations, the *JSON Schema*[11] library will be used. This library allows users to create a schema and validate a JSON object according to it. It provides this functionality for multiple languages, making it ideal for dealing with parameterization across different algorithms and their implementations. This type of schema means that parameters can be easily validated against the JSON Schema file provided by

[7]Found at `https://www.pulumi.com/`
[8]Found at `https://www.ansible.com/`
[9]Found at `https://yaml.org/`
[10]Found at `https://skaffold.dev/`
[11]Found at `https://json-schema.org/`

each algorithm. Along with the parameters, it is also possible to know the output of each algorithm using this library, making it very useful in these two different scenarios.

One of the possibilities of handling the different algorithms is the use of Serverless computing [42], more specifically Functions-as-a-Service (FaaS). This service allocates machine resources according to the demand, allowing inherent scalability. Despite its name, Serverless computing still works with servers but abstracts the idea by providing the user with a way to add functionalities and then resolving the server allocation in its backend [11]. With this type of service, it is possible to receive the algorithms and make them available to be run by the users. However, with this approach, the Serverless module is responsible for assigning and allocating the servers, which can lead to the overuse of resources if not used carefully. Another problem with this type of service is that the functions provided must be distinguishable from one another, meaning each function should have its own identity, enabling it to be called.

4.4 Backend

The development of the backend of the TRUST-AI platform can be done using many different languages, such as Java[12], JavaScript[13] / Typescript[14], PHP[15], and Python[16]. However, due to the time constraints of this thesis's work the TypeScript language was chosen. The TypeScript coding language "is a strongly typed programming language that builds on JavaScript" [46]. Where as, "JavaScript is a lightweight, interpreted programming language with first-class functions" [36], mostly used in Web Pages. This means that TypeScript possesses the same functionalities of JavaScript, while also providing a strongly typed environment that is less prone to developer errors, which speeds up the development time of the backend service.

To use TypeScript for backend development we use Node.js[17], an asynchronous event-driven runtime for JavaScript, that is designed for scalability. With Node.js, we also use a package manager called npm[18], a software registry used to share and borrow JavaScript and TypeScript packages. This allows us to install the "typescript" package and compile our TypeScript code into runnable JavaScript code.

4.4.1 Server

For the backend to receive HTTP requests, it must have an open port for them. However, this is insufficient to have a fully functional server capable of receiving multiple requests simultaneously. If only one port is available, only one connection can be established for these requests, which is not a scalable approach to the backend of a platform. For this reason, we must use a framework

[12]Found at `https://www.java.com/`
[13]Found at `https://www.javascript.com/`
[14]Found at `https://www.typescriptlang.org/`
[15]Found at `https://www.php.net/`
[16]Found at `https://www.python.org/`
[17]Found at `https://nodejs.org/en/`
[18]Found at `https://www.npmjs.com/`

that already solves this issue since solving it ourselves would take away resources from the main objective of this thesis. Among the available frameworks, the following were considered:

- **Express.js**[19] - Express.js does not have a steep learning curve, instead, it requires a basic understanding of Node.js and knowledge of the JavaScript language. It is fast, robust, and asynchronous which allows multiple operations to be executed independently of each other. It also possesses HTTP helpers, which make programs more intelligible. Express.js uses a Model-View-Controller architectural pattern, which allows for multiple types of applications to be designed with it.
- **Koa.js**[20] - Koa.js allows the creation of different web services, by efficiently handling HTTP middleware functions in a stack-like method. This framework is similar to Express.js in terms of flexibility while also giving the developer more freedom. Despite this, the Koa.js framework is still being developed and has a small growing community.
- **Meteor.js**[21] - Meteor.js is an open-source, JavaScript web framework. It is also cross-platform and allows for rapid prototyping with CLI commands. It is written in modern JavaScript code, making it very efficient. This framework is full-stack, meaning it also sends data to be rendered by the user.

Of the frameworks mentioned above, we chose Express.js due to its strong community, which helps support the development whenever an error appears, and because it is a lightweight, backend-focused framework. This framework allows us to configure multiple endpoints using the Express.js Router object. These endpoints use middleware functions that provide a service or capability, for instance, parsing the request's data or validating it.

4.4.2 Libraries

As we are using Express.js we need to use some external libraries to provide some added security and functionality to the backend. The following libraries were installed using npm:

- **body-parser**[22] - The communication with the frontend service is done exclusively through JSON data structures. As such, to process and quickly obtain this data into a TypeScript object, in the backend Application Programming Interface (API) endpoints we use the *json* middleware function present in the library.
- **cookie-session**[23] - As is the case with many platforms there is a need to store information about the client, to guarantee that we can provide some functionalities, for instance, authentication. To do this we use the cookie-session library which provides a middleware function that stores session data not server-side, but client-side within a cookie.

[19]Found at https://expressjs.com/
[20]Found at https://koajs.com/
[21]Found at https://www.meteor.com/
[22]Found at https://www.npmjs.com/package/body-parser
[23]Found at https://www.npmjs.com/package/cookie-session

- **cors**[24] - By using the cors npm library we can use Cross-Origin Resource Sharing (CORS) to allow resources to be requested from another domain, allowing our backend to be integrated with any application that should choose to do so.
- **Express Validator**[25] - The Express.js framework does not provide methods of verifying a request's data. Instead it allows us to define middleware functions that can validate them. However, the validation process can be difficult and time-consuming to implement. As such we use the Express Validator library which provides the necessary tools to define these middleware functions with pre-built and highly configurable validation functions.
- **ExpressJS Async Errors**[26] - The Exppress.js framework has a flaw when using asynchronous middleware functions, which happens whenever there is an error that is not locally caught in the middleware function, stopping the entire server, instead of being sent to a middleware. Because of this, we need to use the ExpressJs Async Errors in order to have a dedicated middleware function for error catching, avoiding unnecessary backend crashes.
- **Multer**[27] - Since we are expecting to receive files from users we need a way of handling file uploads. To do this we use the Multer library, that provides a middleware for handling `multipart/form-data`, which is the protocol used by HTML for uploading files. It also gives us the option of processing and validating the files received, which allow us to have an extra security layer for file uploads.

4.5 Data Storage

Since we use JSON data structures not only in the backend for the algorithms' configuration but also for communicating with the interface, in order to not have to parse this information into another data structure, we use the *MongoDB*[28] database to store TRUST-AI's data. MongoDB is a non-relational database that uses JSON documents to store information. For this reason, MongoDB reduces development time by allowing straightforward storage of the data structures used. This database also allows the storage of the datasets and any other file with the GridFS[29] specification, therefore, storing all the information that TRUST-AI manages. Since this database is open-source, we can deploy it in our own environment/server, which is a key requirement for the database of the TRUST-AI platform since the data stored in the platform can be sensitive. This is the case of the paraganglioma cancer detection use case. For security reasons, this data may be required to be stored in a privately owned server, which can be done with the MongoDB database.

[24]Found at `https://www.npmjs.com/package/cors`
[25]Found at `https://www.npmjs.com/package/express-validator`
[26]Found at `https://www.npmjs.com/package/express-async-errors`
[27]Found at `https://www.npmjs.com/package/multer`
[28]Found at `https://www.mongodb.com/`
[29]Found at `https://www.mongodb.com/docs/manual/core/gridfs/`

Chapter 5

Platform Implementation

The development of the TRUST-AI platform followed an Agile methodology [7] allowing us to follow and track both functional and non-functional requirements. This way, we can better define breaking points in the development of TRUST. It also allows us to continuously design and improve the system, focusing on having a functional prototype throughout the development instead of a collection of functionalities. The development phase was divided into five steps:

1. Implementation with single predictive algorithm
2. Generalization of platform for predictive algorithms
3. Dynamic predictive algorithms and scalability
4. Introduction of prescriptive algorithms
5. Generalization and dynamic prescriptive algorithms

For this effect, the development cycle was divided into sprints, each lasting seven days. At the end of these, there was a meeting to discuss the week's development progress and the next steps in the TRUST-AI implementation.

With this distribution of work, we planned to have a working prototype at the end of the first sprints. From this point, the development of the TRUST-AI platform should always be functional, making it easier to keep track of the development and new features. The sprints start with the implementation of the TRUST-AI platform with a single algorithm making it easier to understand and develop the backend for the frontend functionalities. From this point onward, the goal is to generalize the platform so that different algorithms can run, first, with a static approach where the platform gets its algorithms from the project's source, meaning static algorithms. Then, with an understanding of how these algorithms are set up in the infrastructure, we move the focus to a more dynamic approach of asking the user for a new algorithm to run, changing the infrastructure on demand. These first algorithms implemented in the TRUST-AI platform were suited to solve predictive problems, as they are more straightforward in terms of functionality and results when compared to prescriptive algorithms (which often include a simulation engine that is problem specific). In the next sprints, we add a simple prescriptive algorithm and implement the prescriptive functionalities that come with it. After this point, and with the knowledge of the predictive implementation, we implement both the static and dynamic acquisition of these algorithms.

In this chapter, we start by defining the architecture of the TRUST-AI platform in Section 5.1. Next, we discuss the use of Kubernetes in the platform and how the backend manages it in Section 5.2. We then detail the algorithm acquisition process and the steps required to add an algorithm to the platform in Section 5.3, along with the limitations and definitions imposed by the platform on the algorithms Section 5.4. We also detail the REST API created in the backend in Section 5.5, as well as the database used and the data there stored in Section 5.6. Finally, we describe the proposed communication method with the Cognitive Models component, which is planned for the TRUST-AI platform, and its implementation in the backend in Section 5.7.

5.1 Architecture

The platform was designed by taking advantage of containers. The diagram 5.1 shows the full architecture of the platform, which is divided into six components, namely: Load Balancer; Web Interface (out of the scope of this thesis); TRUST Backend; Database; Job Queue; and Algorithm Runs.

Since TRUST-AI is an online platform, requests are the primary communication method. These originate in the user's browser and arrive at the Load Balancer component of the TRUST-AI. This component is responsible for redirecting the request to the correct service, which is either the Web Interface or TRUST-AI Backend.

The TRUST-AI Backend component has two different responsibilities. The first is the storage and retrieval of data in the Database, which is done by the API module. The second is tracking the algorithm runs through the Job Manager module to obtain their results. An algorithm-run Job can be requested using the API module, which is passed to the Job Manager module for processing. This module then creates a new Job in the Job Queue component that, by tracking the resources available in the platform, either starts the algorithm run or puts it on hold until these resources are free.

The Algorithms component does not actively communicate with any of the other modules. Instead, it must be queried for data, which should not be the Algorithms' responsibility.

As we have algorithms that produce data along with users that wish to store and query data, we must have a Database component. This component should only interface with the TRUST-AI Backend, as it is responsible for communication with the user and the algorithms.

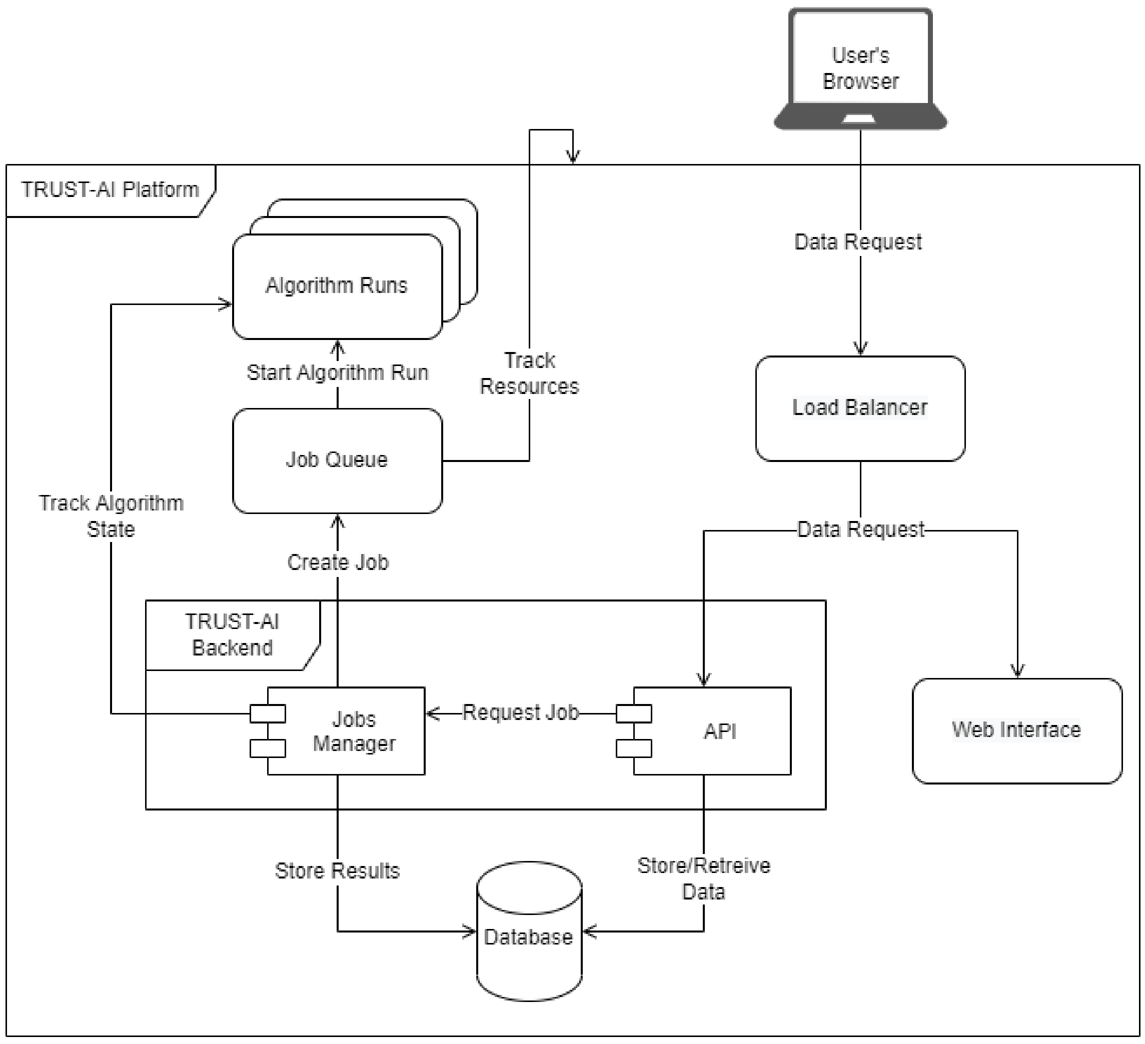

Figure 5.1: Architecture of the TRUST-AI platform.

5.2 Kubernetes

The architecture, as seen in Figure 5.1 was implemented in Kubernetes, specifically inside a Kubernetes cluster. A Kubernetes cluster is a grouping of nodes that runs containerized applications, which allows the platform to scale horizontally as long as there is a Load Balancer. The Load Balancer class in Kubernetes is responsible for receiving requests and transmitting them to a service with the resources to respond. The apps that run in a Kubernetes cluster use Docker containers. Each app runs in a Kubernetes Pod, which can have multiple containers, each running a different Docker image. However, suppose a Pod needs to be accessible without using the Kubernetes API, which needs special permissions. In that case, a Kubernetes Service needs to be created for this specific Pod, which describes a port to which requests can be sent to.

The TRUST-AI platform uses a Load Balancer called Ingress NGINX, which can be used to set up rules for the requests received so that the API requests can be sent to the backend service while

```yaml
- host: user.trust.dev
  http:
    paths:
      - path: /api/?(.*)
        pathType: Prefix
        backend:
          service:
            name: trust-srv
            port:
              number: 3500
      - path: /?(.*)
        pathType: Prefix
        backend:
          service:
            name: users-web-srv
            port:
              number: 3000
```

Listing 5.1: Ingress rules defining the distribution of the HTTP requests received.

having the frontend service under the same domain. This is done via rules, which can be seen in detail in Listing 5.1. These define that all requests to the “user.trust.dev” domain whose path starts with “/api/” will be sent to the “trust-srv” service, which is the backend service of the TRUST-AI platform. It also defines another rule for the same domain that accepts any request, meaning that if the request is not for the backend, it will be directed to the frontend service called “users-web-srv”. Without Ingress, the communication with the backend service would have to be done using the URL “http://trust-srv:3500/api/...”, since the way to access a Kubernetes Service is to use its service name, in this case “trust-srv” and then the port where the Node.js server is running. This is extremely precise and should be avoided since it requires users to know the exact service name and port. Also, with this configuration, the frontend and backend would have different URLs, which is undesirable. The Ingress also stores a Secure Sockets Layer (SSL) certificate to allow for HyperText Transfer Protocol Secure (HTTPS) requests, providing secure communication with the services. Even if a certificate is not provided, an unsigned SSL certificate is generated, which can be used in development environments for HTTPS communication with the server.

The Kubernetes is designed to be secure by default. This is achieved by authenticating and authorizing requests to its API according to Role Based Access Control (RBAC) policies. As such, access to the Kubernetes API in the backend of TRUST-AI requires some permissions, which is done by creating a Kubernetes Role and assigning it to the backend. These include access to pods and jobs, with execution and logs acquisition permissions. This allows the backend service to start new jobs and copy files from the pods to obtain results from algorithm runs. With this, the platform can fully control the execution of the algorithms, including checking the status of the algorithm, starting the run, getting the results and logs, and terminating runs.

5.2.1 Algorithm Run

For the platform to run an algorithm, it uses Kubernetes Jobs. These require two specifications, one for the Job itself and another for the container that will run the actual algorithm. These specifications are detailed in Listing 5.2, where the YAML file representing the Kubernetes Job is listed.

The Job specification requires a unique name, described in the configuration above in its metadata as `<unique-name>`, for which we use the session ID. We also define metadata that allows the platform to know the session for which the Job is running. In this specification, we also set the `backoffLimit` to "0" and `restartPolicy` to "Never" so that if the algorithm fails, a new container is not created to try to rerun it.

The container specification, found after line 12 in Listings 5.2, requires an image, which is obtained from the algorithm specification. It also requires the specification of the minimum amount of resources the algorithm needs via the `resources` object. The specification then configures the Job's command in the Docker image. It is in this command that several aspects of the algorithm execution are controlled. These are divided into the following four stages:

1. **Create the results directory** to which the results will be saved using a simple `mkdir` command. The directory is also sent via an environment variable called `RES_DIR` so that the algorithm can know to which directory to send the results.

2. **Download the dataset and configuration files**, which were chosen by the user, can only be downloaded through a private API only available from within the platform's Kubernetes cluster. This is done by using the `curl` command.

3. **Run the algorithm**, where the command to run the algorithm is called, just as it is specified in the algorithm configuration database entry.

4. **Notify backend of algorithm termination**, when the algorithm run has finished, meaning we should acquire the latest results and logs. For the platform to do this without the Kubernetes Job reaching the "Succeeded" status, triggering the deletion of the Kubernetes Pod's containers which has the algorithm results. A job termination route is called, which forces the container to wait for the route's response before termination, ensuring that the container is not deleted, along with the results. The route calls the Kubernetes Module acquiring the results, logs and, afterwards, terminating the Kubernetes Job.

This setup is sufficient for the successful execution of the algorithm and the acquisition of its data. However, an algorithm execution can fail, whether through an error in the algorithm's code, a wrong dataset being used, or an invalid configuration being passed.

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: <unique-name>
  labels:
    session: <sessionID>
spec:
  backoffLimit: 0
  template:
    spec:
      restartPolicy: Never
      containers:
        - name: trust-ai-algorithm
          image: <algorithm-image>
          env:
            - name: RES_DIR
              value: '/results'
          imagePullPolicy: IfNotPresent
          resources:
            requests:
                cpu: <value>
                memory: <value>
          command:
            - '/bin/sh'
            - '-c'
            - 'mkdir \$RES_DIR &&
               curl -s -o ./dataset
               http://trust-srv:3500/api/
                 algorithm-run/dataset?token=<JWT> &&
               curl -s -o ./config.json
               http://trust-srv:3500/api/
                 algorithm-run/configuration?token=<JWT> &&
               <entrypoint>  &&
               curl -s --keepalive-time 300
               http://trust-srv:3500/api/
                 algorithm-run/finish?token=<JWT>'
```

Listing 5.2: Kubernetes Job specification of a algorithm run.

5.2.2 Algorithm State Control

In order to control the state of the algorithm run, an asynchronous task is launched every minute using the *toad-scheduler*[1] library. This task acquires the current running Jobs in the Kubernetes cluster, checks their status, and executes actions accordingly. Each one of these Jobs can be in one of the following statuses:

- Succeeded - The Job has terminated with a status code of 0, as such, the final logs can be saved and the Job can be deleted.
- Failed - The Job has terminated with an error status code, as such, the logs can be saved and the Job can be deleted.
- Running - The Job is still running. With status we retrieve the current results and logs to allow users to track the progress of the algorithm run.
- Pending - The Job is in queue to be launched.
- Terminating - The Job is terminating after being tagged for deletion by the platform.
- ErrImageNeverPull - Kubernetes did not find the image that was requested by the algorithm. This status code is similar to the Failed status but there are no logs to be obtained.
- ContainerCreating, Unknown - The Job is at a stage where nothing should be done.

The platform ignores any status that is not present in the list above. Also, since this module is event-based, there is a possibility of having race conditions between the algorithm state control module and the algorithm termination stage. Therefore we introduced a mechanism to control the access to the database's documents, specifically to the document that stores the information regarding an algorithm run. This mechanism is based on the principle of semaphores that allow or stall access to a resource. To develop such a mechanism, we used an npm library called mongo-locks[2] to simulate semaphores with the help of the MongoDB database.

5.2.3 Data Acquisition

As the algorithm runs, it starts producing data that needs to be passed on to the platform for storage and processing. This process, which is done only at certain stages as described in section 5.2.2, relies on the Kubernetes API. More specifically, it uses the `logs` command to retrieve the logs of the algorithm and an altered version of the `cp` command to obtain the results files.

However, since `cp` calls the `tar` command internally, it is a requirement that the algorithms' docker image have the `tar` program installed, which collects several files into one archived file. Should they not have it, the algorithm run will always terminate with an error code as the last command will call the `cp` command from the platform's backend service.

[1]Found at `https://www.npmjs.com/package/toad-scheduler`
[2]Found at `https://www.npmjs.com/package/mongo-locks`

After the files are copied into the platform, each file is stored individually in the database. It also processes the files should they be defined by the framework of the TRUST-AI platform. These files are be discussed in Section 5.4, more specifically in Section 5.4.2.

5.3 Algorithm Acquisition

In order to run an algorithm, it first needs to be available to the platform. However, since we are using Kubernetes and want to isolate each component as much as possible, the algorithms must run in a Kubernetes Job. As such, all algorithms in the platform are received and used in the form of Docker images during the platform run-time. These images are then stored in a private Docker repository, accessible to the Kubernetes cluster. To send these algorithms to the platform, the algorithm developer must use one of the two possible ways:

- **Through project files** (static configuration) - the algorithm code is inserted into a specific project folder for algorithms. It is then configured to be compiled using the Skaffold configuration file when launching the platform. Moreover, its configuration schema and other properties such as image name and description are introduced via JSON file in a specific folder of the TRUST backend service.
- **Importing Docker image** (dynamic configuration) - this option allows the user of the platform to introduce a new algorithm while the platform is running. This is done via uploading a zip/tarball file containing the algorithm code, along with a Dockerfile responsible for the compilation of the algorithm into a docker image. This compressed file is then used to build a docker image using the dockerode[3] npm library, which interfaces with the docker socket present in the backend service. For this effect, we are not running a Docker service in the backend. Instead, we are using a Docker-out-of-Docker (DoD) approach where the host's Docker socket is being sent to the backend service, giving it full permissions to access Docker. Along with the compressed file, the user must also give additional information regarding the algorithm, including a brief description and the configuration schema for the parameters the algorithm can receive.

The static configuration of the algorithms allows the developer to have the algorithms available on platform startup. This also means that any change in the algorithm parameters requires either a reboot of the backend so that the algorithm configuration JSON file is sent to the database or that the developer accesses the MongoDB database manually and alters the algorithm document. The reason is that these algorithms do not have an owner during runtime. Furthermore, a change in the algorithm configuration JSON file likely means that the algorithm was also changed, requiring it to be recompiled by the platform, which is only done on platform launch. However, due to the use of Skaffold for the development environment, this issue is not very relevant since it detects differences in the files and automatically sends them to the backend service, which is running in

[3]Found at `https://www.npmjs.com/package/dockerode`

```
{
  "name": "Name of the algorithm",
  "description": "Description of algorithm",
  "type": "predictive/prescriptive",
  "entrypoint": "command",
  "secret": {
    "cpu": 1,
    "memory": 1
  },
  "configuration": {
    (JSON_Schema)
  }
}
```

Listing 5.3: JSON data structure required to add algorithms to the TRUST-AI platform.

dev mode, meaning it recompiles and restarts the server whenever there are code changes. In a production environment, this option should be only used for algorithms that are not subject to change since it would require a restart of the backend service.

The dynamic configuration feature allows users to run their specific algorithms on the platform. This feature has a security issue since we are using Docker-out-of-Docker (DoD), which has many security issues since it gives root permission to the host's Docker.

5.3.1 Configuration

Independently of the method used to introduce the algorithm in the platform, the Algorithm Developer must always provide some specifications using a JSON data structure, regarding not only the parameters that the algorithm receives but also the minimum resources needed to run the algorithm. The full extension of the data passed is presented in Listing 5.3.

This configuration allows the Algorithm developer to describe the algorithm using the name, description, and type of algorithm. This type can only be either predictive or prescriptive, as this will allow Model Developers to better understand the algorithm. The `entrypoint` parameter in the JSON structure is the command that will start the execution of the algorithm. Along with this, `cpu` is the number of CPUs required by the algorithm, and `memory` is the number of GB of memory required.

The parameters received by an algorithm can be as trivial as the definition of the number of individuals in each population to the definition of the terminals and operators to use in the run. For this reason, we decided to use JSON files to convey these configurations to the algorithms. Working with JSON data structures is also trivial in Javascript and Typescript and is already used to pass data from the frontend to the backend of the platform. Furthermore, JSON is a commonly used data-interchange format, meaning most, if not all, coding languages support JSON data.

However, with JSON files as configurations, we still need to have a way of defining the structure of the data passed to the algorithm. To solve this problem, we used the JSON Schema[4] specification. More specifically, we used the AJV[5] library, which implements the JSON Schema concepts and functionalities and adds some extra features which are useful for the definition of the parameters, for instance, the default value of a parameter. When adding the algorithm to the platform, the algorithm developer only needs to introduce the JSON Schema of the parameters. The platform will validate the parameters input by the users when setting up their algorithm run. This way, the parameters received by the algorithm always have the correct data format.

As such, the "configuration" parameter in the JSON file received along with the algorithm is reserved for the JSON Schema that will validate the parameters. This schema is shown to Model Developers along with the algorithm's name, description, and type, allowing them to know the full extent of parameters available in the algorithm.

The configuration shown in Listing 5.3 is enough for the dynamic insertion of the algorithm. However, a unique ID must also be given for the static approach. This ID is used to ensure the same algorithm configuration is not added twice to the database and is updated when the platform restarts.

5.4 Algorithm Requirements

In this section, we discuss the implications of the platform's implementation on the algorithms and the requirements that these must obey to correctly receive the users' files and parameters, as well as output the generated data to the platform.

Since these algorithms can have different configurations, ways of representing data, and outputting the said data, a framework had to be developed to have some common ground between these algorithms. This framework does not impose a language of development. Instead, it defines how to pass different parameters and files to the algorithms, how to output results and some additional programs that must be installed in the algorithm's Docker image. This information is also detailed in the project's folder, as it should be readily available to algorithm developers.

5.4.1 Algorithm Parameters and Dataset

The parameters defined by the user and validated with the JSON Schema file provided by the algorithm are sent to the algorithm via an HTTP request to the platform and placed in the main directory of the algorithm's container under the name of *config.json*. The process for the dataset is the same, and it is saved under the name of *dataset* in the same folder. However, there is no validation of the file being sent by the user. This can be both a positive and negative aspect, as on the one hand, the algorithm can receive any file, even zipped folders, while on the other hand, the type of file and its contents might cause an error in the algorithm. Moreover, even though an

[4]Found at `https://json-schema.org/`
[5]Found at `https://ajv.js.org/`

error should be avoided, in this case, it can be seen as a cause of the level of abstraction that the platform provides, as we do not limit the algorithms to any type of dataset files.

The HTTP request to download the parameters and dataset requires the usage of the `curl` program, which should be installed in the docker image. If the program is not present, the algorithm will always produce an error when it is attempted to run.

5.4.2 Algorithm Results

The resulting files of an algorithm run can follow different formats and be saved in different directories. For this reason, we decided to introduce a standard that all algorithms of the platform must comply with, should algorithms output data besides logs. The standard defined is that for an algorithm file to be saved, it should be stored under the directory listed in an environment variable called `RES_DIR`. This variable is specified through Kubernetes in the algorithm's container specification.

Furthermore, the TRUST-AI platform was developed with the main focus on GP algorithms meaning that the output of the majority of the algorithms is in the form of symbolic expressions. As such, if some files have the name and structure that is expected of a GP algorithm, they should be parsed and processed by the platform into usable data by the frontend. This process happens when the platform acquires data from the algorithm run, and the expected formats are as follows:

- Generations - a file named *generations.csv* (Table 5.1) that has the information regarding the best expression found in each generation of the GP run.

Table 5.1: Generations file structure (generations.csv)

Generation	**Model**	**Fitness**	**Size**	**<extra-1>**	**...**	**<extra-n>**
1	*<model-1>*	*<fitness-1>*	*<size-1>*	*<extra-1-1>*	...	*<extra-n-1>*
⋮	⋮	⋮	⋮	⋮	⋮	⋮
k	*<model-k>*	*<fitness-k>*	*<size-k>*	*<extra-1-k>*	...	*<extra-n-k>*

- Populations - a group of files named *population_X.csv* (Table 5.2), where the X is the number of the generation to which the population belongs to. Thus, each file contains the list of expressions of the generation and their data.

Table 5.2: Population file structure (population_X.csv)

Individual	**Model**	**Fitness**	**Size**	**<extra-1>**	**...**	**<extra-n>**
1	*<model-1>*	*<fitness-1>*	*<size-1>*	*<extra-1-1>*	...	*<extra-n-1>*
⋮	⋮	⋮	⋮	⋮	⋮	⋮
k	*<model-k>*	*<fitness-k>*	*<size-k>*	*<extra-1-k>*	...	*<extra-n-k>*

- Metrics - a file named *metrics.csv* (Table 5.3) that has any metric that the algorithm might export that is not related to a single expression but to the run itself.

Table 5.3: Metrics file structure (metrics.csv)

<metric-header-1>	**...**	**<metric-header-n>**
<metric-1>	...	*metric-n*

With these formats, we can obtain any data the algorithms might output relative to GP expressions and use it in the frontend. This also means that it is the responsibility of the algorithm developer to adapt his algorithm to the platform. However, since the outputs specified are very common in GP algorithms, the effort to do so is expected to be minimal. The formats for these results are also modular, ensuring that any changes in framework formats can be quickly implemented in the platform, which can be done both for new files or the existing ones.

5.5 REST API

A REST API was developed for the TRUST-AI platform in the backend service as a means of communication between the backend and frontend. The API was created not only as an interface between the frontend and the database but also to launch algorithm runs by communicating with the Kubernetes Module. Each API endpoint validates its inputs via the validation middleware created using the library "express-validator". Along with input validation, we also developed other middleware functions that enforce authorization rules, for instance, ensuring that the user is authenticated and that the project they are accessing is their own. All endpoints use the same final middleware to capture any Typescript errors that might have occurred in the processing of the request. Whether the error was purposely created by enforcing rules, in which case an error response such as in Listing 5.4 is sent, or generated by a bug in some part of the code. Should the latter occur, the error is logged in the console while generating a response with a formatted message as shown in Listing 5.5. The error response message format is always the same, allowing easier processing of the error by the frontend application.

The endpoints available in the REST API are shown in Table 5.5. Each endpoint is configured to use at least one of four HTTP methods: GET, POST, PUT, and DELETE. Three of the methods used are idempotent, meaning that multiple calls with the same information will result in the same response from the API. These methods are: GET, PUT, and DELETE. The definitions of the methods are as follows:

- **GET** - Requests the information of the specified resource.
- **POST** - Submits new data to the specified resource.
- **PUT** - Changes the data of the specified resource.
- **DELETE** - Deletes the specified resource.

```
{
    "errors": [
        {
            "message": "Invalid value",
            "field": "name"
        },
        {
            "message": "name must not be empty",
            "field": "name"
        },
        {
            "message": "type must be one of : predictive,
                        prescriptive, any",
            "field": "type"
        }
    ]
}
```

Listing 5.4: Example of an error response message for the creation of a project.

```
{
    "errors": [
        {
            "message": "Something went wrong"
        }
    ]
}
```

Listing 5.5: The response message sent whenever an internal error occurred in the platform.

Table 5.4: Routes defined in the REST API of the TRUST-AI platform. Endpoints that contain a word that starts with ‘:’ represents a variable whose value can only be known in run-time.

Name	**Method**	**Endpoint**	**Description**
Current User	GET	/api/users/currentuser	Provides information regarding whether the user is logged in or not
Sign In	POST	/api/users/signin	Allows the user to authenticate himself in the platform
Sign Up	POST	/api/users/signup	Allows the user to create an account in the platform
Sign Out	POST	/api/users/signout	Signs out the user, removing access to platform until new sign in
Get Algorithms	GET	/api/algorithms	Shows the list of algorithms available in the platform
Add Algorithm	POST	/api/algorithms	Adds a new algorithm to the list of available algorithms
Get Algorithm By ID	GET	/api/algorithms/ :algorithmId	Shows the information of the algorithm whose ID is “algorithmId”
Edit Algorithm	POST	/api/algorithms/ :algorithmId	Changes the stored information regarding the algorithm whose ID is “algorithmId”
Get Projects	GET	/api/projects	Shows the list of projects created by the user
Get Project By ID	GET	/api/projects/:projectId	Shows the information of the project whose ID is “algorithmId” if it belongs to the user
Create Project	POST	/api/projects	Creates a new project with the data provided
Get Datasets	GET	/api/projects/:projectId/ datasets	Shows the list of datasets that were uploaded to the project
Send Dataset	POST	/api/projects/:projectId/ datasets	Stores the file received in the platform
Get Dataset By ID	GET	/api/projects/:projectId/ datasets/:datasetId	Shows the information of the dataset whose ID is “datasetId”
Delete Dataset	DELETE	/api/projects/:projectId/ datasets/:datasetId	Removes the specified dataset from the platform
Get Dataset Data	GET	/api/projects/:projectId/ datasets/:datasetId/data	Show the data present in the dataset file sent

Continued on next page

Table 5.4 – continued from previous page

Name	**Method**	**Endpoint**	**Description**
Download Dataset	GET	/api/projects/:projectId/ datasets/:datasetId/ download	Downloads the data present in the dataset file sent
Get Sessions	GET	/api/projects/:projectId/ sessions	Shows the list of sessions that belong to the project specified
Create Session	POST	/api/projects/:projectId/ sessions	Creates a new session that belongs to the project specified
Get Session By ID	GET	/api/sessions/:sessionId	Shows the information of the session whose ID is “sessionId”
Update Session	PUT	/api/sessions/:sessionId	Changes the stored information of the session whose ID is “sessionId”
Get Algorithm Schema	PUT	/api/sessions/:sessionId/ schema	Shows the schema of the algorithm chosen in the session
Get Configuration	GET	/api/sessions/:sessionId/ configuration	Shows the current configuration set in the session
Set Configuration	POST	/api/sessions/:sessionId/ configuration	Adds or updates the configuration for the algorithm chosen in the session
Run Algorithm	POST	/api/sessions/:sessionId/ run	Launches an algorithm run for the session
Get Logs	GET	/api/session/:sessionId/ logs/data	Shows the logs generated by the algorithm run for the specified session
Download Logs	GET	/api/session/:sessionId/ logs/download	Downloads the logs generated by the algorithm run for the specified session
Get File Data	GET	/api/session/:sessionId/ files/:fileId/data	Shows the contents of the results file whose ID is “fileId’
Download File	GET	/api/session/:sessionId/ files/:fileId/download	Downloads the results file whose ID is “fileId’
Get Generations	GET	/api/session/:sessionId/ results/generations	Retrieves the generations data that was processed from the algorithm run’s output
Get Metrics	GET	/api/session/:sessionId/ results/metrics	Retrieves the metrics data that was processed from the algorithm run’s output

Continued on next page

Table 5.4 – continued from previous page

Name	**Method**	**Endpoint**	**Description**
Get Populations	GET	/api/session/:sessionId/results/populations	Retrieves the all the populations data that was processed from the algorithm run's output
Get Population	GET	/api/session/:sessionId/results/populations/:generation	Retrieves population whose generation is "generation" from the data that was processed from the algorithm run's output
Save Result	POST	/api/session/:sessionId/results	Saves the specified result in the session's results
Get Saved Results	GET	/api/session/:sessionId/results	Retrieves the results saved by the user
Delete Saved Result	DELETE	/api/session/:sessionId/results/:resultId	Delete the result whose ID is "resultId" from the saved results

5.5.1 Authentication

The authentication process in the TRUST-AI begins with account creation, where the user defines his email, first and last name, and password. The email and password are then required every time the user wishes to access the platform, therefore, requiring the user's password to not be easily discoverable by enforcing a few rules and also be stored securely. The rules enforced in the password are a minimum of 6 characters, one uppercase, one lowercase, and one digit, without spaces, while also filtering some of the more common passwords, as is the case of "Password123". The password is then encrypted before its storage in the database. The encryption method used for this effect is called *scrypt*[6], and along with it, we use a random string, commonly designated as salt, of 64 characters. This way, even if two users have the same password, they will be stored with a different value, making it harder for an attacker to know a user's password.

The authentication and identification of users in the API after logging in is made using JSON Web Tokens[7] (JWT). A JWT is created during user authentication and sent to the user's browser, where it is stored in the session. This JWT contains some information that identifies the user in the platform, as seen in Table 5.5. The information is placed in a JSON object and is then encrypted using a key that we define as a secret in the Kubernetes Cluster and pass to the backend in its Kubernetes definition. The JWT should always be sent to the endpoints the user is accessing, which is automatically done by most browsers. It is then verified in every route via a middleware that attempts to decode the JWT. This process is secure because if there has been any tampering with the JWT, the decoding process will fail, giving an error code of 401.

[6]Found at `https://www.rfc-editor.org/info/rfc7914`
[7]Found at `https://jwt.io/`

Table 5.5: Data stored in a JWT sent to the User upon authentication

Name	**Type**	**Description**
ID	MongoDB ID	The id of the user in the database
Email	String	The email of the user
Language	String	The language that the user chose for the platform

5.5.2 Frontend Integration

During the platform's development, the frontend was also being developed in [22], which required constant communication with the frontend developer to make sure features requested in the interfaces could be implemented with the help of the REST API. This communication was primarily done via meetings. However, the API implemented in the backend had to be visible to the outside for faster development. For this, we used Postman[8] which not only publishes the API but also allows users to call it, speeding up the development of both backend and frontend. The final Postman collection can be seen in Postman by accessing the TRUST-AI public workspace[9].

5.5.3 Private API

The private API that was developed to control the algorithm also uses JWTs to validate the requests. However, the data stored in the JWT is different as the algorithm run does not belong to a user but to a session. In Table 5.6 the information stored in the algorithm JWT can be seen. With this information, even if an algorithm had malicious code that would attempt to access another route, it would not be able to do so, as the user authentication middleware would not accept the information. The routes created for the Private API can be seen in Table 5.7.

Table 5.6: Data stored in a JWT used in the private API

Name	**Type**	**Description**
Session	MongoDB ID	The id of session that controls the algorithm run

Table 5.7: Routes defined for the private API, only accessible by algorithms.

Name	**Method**	**Endpoint**	**Description**
Get Configuration	GET	/algorithm-run/configuration	Downloads the configuration for the algorithm
Get Dataset	GET	/algorithm-run/dataset	Downloads the dataset file that was uploaded to the platform
Finish Run	POST	/algorithm-run/finish	Signals to the platform that the algorithm has finished its execution

[8]Found at https://www.postman.com/
[9]Found at https://www.postman.com/trust-ai/workspace/trust-ai

5.6 Database

The database used by the TRUST-AI platform stores information generated by the user, and the data output by the algorithm runs. As such, the database stores both regular documents and files. The database chosen to store this information was MongoDB, as it saves documents in JSON format, a format compatible with Typescript. MongoDB is a non-relational database, meaning it does not save data in a tabular way, which allows for any JSON data structure to be saved in it, as long as it does not take more than 16MB of space, a limitation imposed by MongoDB. The MongoDB database also has a specification named GridFS that allows storing files via partitioning the data into chunks that are stored separately. The communication with the database was done using an npm library called mongoose[10], which models MongoDB documents into objects in Node.js. With it, we can define a Schema for each type of document and some definitions of the type of variables in Typescript, providing an extra level of input validation. Despite this, mongoose does not provide out-of-the-box integration with GridFS. As such, another library had to be installed, called mongoose-gridfs[11], which integrates the GridFS specification with mongoose. However, this library was not developed for TypeScript but for JavaScript, making development with the library only possible with the help of a reference page. The different documents being saved in the database and their relationships can be found in Figure 5.2.

The usage of mongoose was later found to have an optimization problem. This problem appears when we add an array with thousands of values to a document whose specification of the array is defined, as in the "Session Population" document. Mongoose will validate the input received for each array element, which for four thousand "Individual" objects took about sixty seconds to complete. Since each algorithm run outputs multiple populations, this is a big issue. The solution to the problem is not to define the type of objects received by the array, using "population: Array<Object>" instead of "population: Array<Individual>" while keeping its definition in TypeScript unchanged. This way, the validation process is loosened, allowing for developer mistakes that can occur by casting objects but reducing the strict validation process imposed by mongoose.

5.7 Cognitive Models

The Cognitive Models component was not developed in time to be integrated with the TRUST-AI platform. For this reason, the connection with it was not created, and the component itself was not added to the platform. However, the backend is implemented with tools that allow communication with the Cognitive Models to be developed, more specifically, a framework for communication with any number of services. This framework uses NATS as its core by having a NATS JetStream service running in the Kubernetes cluster, which allows any service to push a message to any other service that is listening to the specific topic of the message [41].

[10]Found at https://mongoosejs.com/
[11]Found at https://www.npmjs.com/package/mongoose-gridfs

TRUST-AI's communication was implemented in the backend using the NATS.js[12] npm library, developed by the creators of NATS. With it, we can connect to the NATS service, listen to topics of a message and publish any message to any topic. Communication was developed this way because it allows the backend to connect to any other service by abstracting the services into message receivers handled by NATS. This way, changes and additions to these services do not directly affect the backend of the TRUST-AI platform. Instead, when a service is updated or replaced by another, it must conform to the messages and topics already present in the system, reducing the number of changes required in the platform.

[12]Found at `https://www.npmjs.com/package/nats`

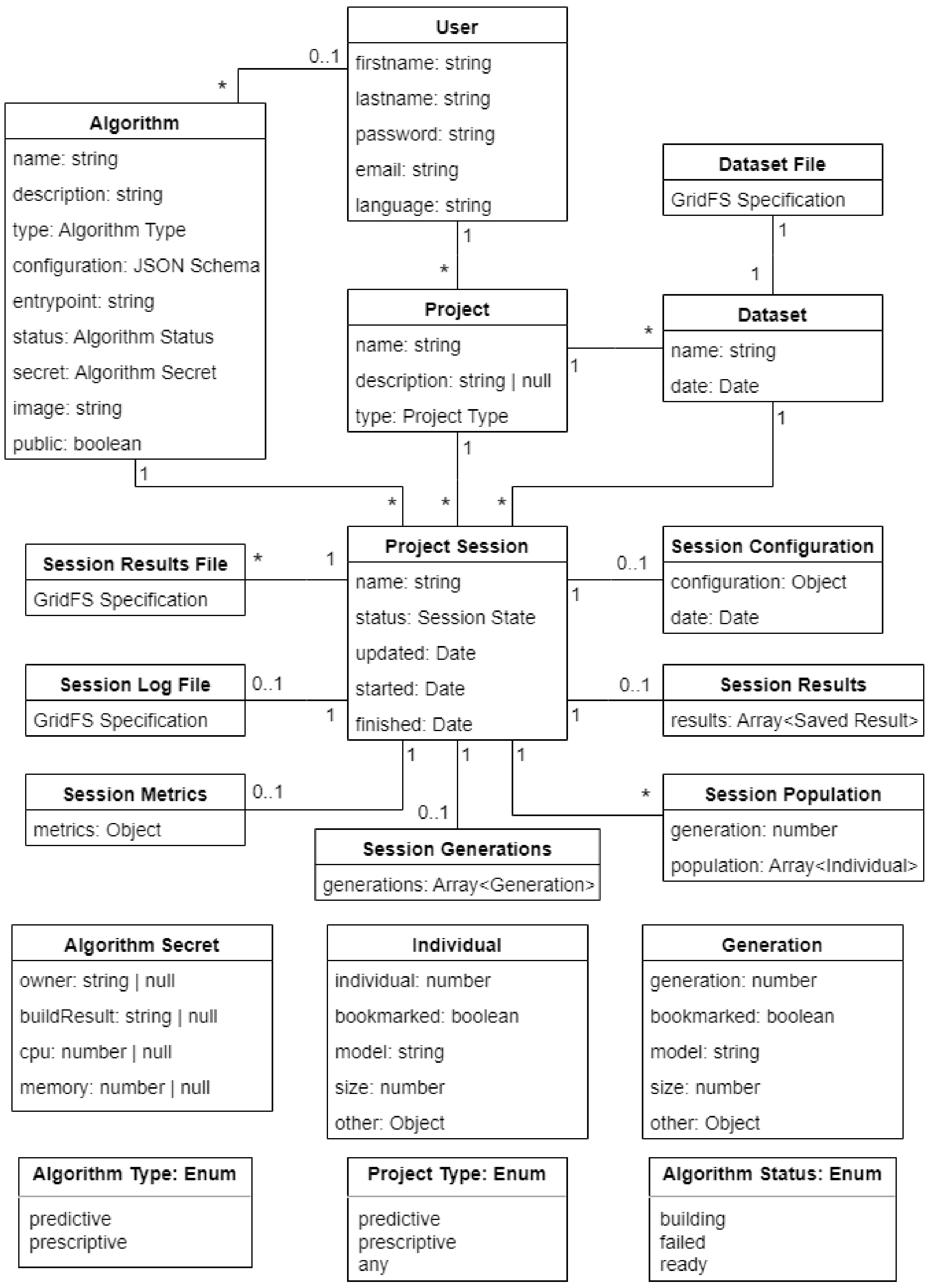

Figure 5.2: Diagram of data structures saved in the database and their relationships.

Chapter 6

Platform Evaluation

After the development of a platform, it must be evaluated to validate the implemented features. For the TRUST-AI platform, performance tests were conducted to verify that the platform conforms to its non-functional requirements while also implementing its functional requirements.

The results presented in this chapter were conducted in a server whose specifications can be seen in Appendix A, inside a VMWare[1] container with MATE Ubuntu[2] running. In this Ubuntu OS, we created a Kubernetes cluster using kind[3] to run the TRUST-AI platform and also execute some performance tests.

In this chapter we first discuss the non-functional and functional requirements proposed for the platform in Sections 6.1 and 6.2, respectively. We then evaluate the platform's ability to use different algorithms in 6.3. Performance tests are also presented and discussed in 6.4. Finally, in Section 6.5 we detail a live demo that was done with the TRUST-AI platform, which includes the feedback related to the overall platform and the backend.

6.1 Non-Functional Requirements

As one of the key factors of the TRUST-AI platform is compliance with its non-functional requirements, these must now be reviewed to validate that they are indeed implemented. The non-functional requirements defined are the following:

- **Performance** - The platform was tested for its performance while running algorithms to validate this requirement. The results of this test can be seen in Section 6.4.
- **Usability** - The platform's usability can be verified by the fact that we were able to introduce five different algorithms to the platform, which can be seen in Section 6.3, as well as the results of the live demo that was done with TRUST-AI partners. The results of this demo

[1]Found at `https://www.vmware.com/`
[2]Found at `https://ubuntu-mate.org/`
[3]Found at `https://ubuntu-mate.org/`

related to the backend can be seen in Section 6.5, while the results related to the interfaces can be seen in [22].

- **Security** - The developed platform possesses features and protocols that enforce security across the whole platform. For user data storage, a secure password encryption algorithm is used, along with an authentication system based on JWTs, which are considered secure. Along with this, the platform also enforces authorization via middleware functions, ensuring that only the right users can access the information.
- **Maintainability** - Since the platform's backend was developed alongside the frontend, maintainability had to be ensured from the beginning of the development cycle. This was done by developing the platform using a micro-service architecture in Kubernetes, where the backend and frontend were developed as separate applications. We also created documentation of the backend in both Postman and the project's GitLab repository. As such, new platform developers have enough information to maintain and further develop the platform.

6.2 Functional Requirements

After the implementation of the platform, we can now review the functional requirements of the TRUST-AI platform, which were proposed in Section 3.1, more specifically in Table 3.1. The following Table 6.1 shows these exact functional requirements, but with the work done on each requirement to either implement it or prepare for the implementation, which is the case of the requirement Model Explanations (R11).

6.3 Algorithm abstraction

As one of the TRUST-AI platform's goals is to be able to run any algorithm, this requirement must be tested. To do so, we tested five different algorithms in the platform, which are described in Table 6.2. The procedure followed to add these algorithms to the platform was the static configuration as described in Section 5.3. A Dockerfile was added to each of these algorithms' source code, which was added to the folder `/algorithms` of the project's files. The algorithms were then added to the `skaffold.yaml` file for compilation on platform start. A JSON configuration file was created for each algorithm with its name, description, and image name as in `skaffold.yaml`, type, algorithm start command, the minimum amount of memory and CPU requirements, and their configuration schema. After these steps were done, the algorithms were run by passing the correct datasets via the platform.

As different algorithms were run in the platform, as shown in Table 6.2, it is possible to conclude that the platform allows algorithms written in different coding languages to be run. The process of adding these algorithms to the platform is also agnostic to the programming language of the algorithm. The Algorithm Developer is expected to write a Dockerfile for each algorithm,

Table 6.1: Final functional requirements of the TRUST-AI platform and the work done.

RID	Name	Done	Work done
R1	Add algorithm	Yes	Receiving algorithm code and compiling it into a Docker image.
R2	Define possible parameters of the algorithm	Yes	Passing the parameters JSON Schema along with the algorithm code.
R3	Parameterize algorithm	Yes	Validating the parameters against the algorithm's JSON Schema.
R4	Upload files	Yes	Having a route dedicated to the upload of files and storing them in the database.
R5	Run algorithm	Yes	Using Kubernetes Jobs to execute algorithm runs, and overriding the algorithm's Docker image command by adding commands to download the configuration and files into the algorithm's code.
R6	Save algorithm data	Yes	Using the Kubernetes API to transfer results and obtain the logs of the algorithm run's Kubernetes Job.
R7	Process algorithm data	Yes	Creating a framework for the algorithms' results and processing only the known files.
R8	Track algorithm run	Yes	Scheduling a task that periodically scrapes the Kubernetes API for Jobs and obtains their data.
R9	Save specific results	Yes	Having a dedicated route and database structure for saving both new results as well as results from an algorithm run.
R10	Run model in algorithm	No	None
R11	Model explanations	No	Designed and implemented the methods of communication with the Cognitive Models component, however this component is not yet implemented.

which can be seen as a cost since, from experience, Docker is not a very common tool for developing these algorithms. Of the five algorithms used, only one (GPG [33]) had an already present Dockerfile. Despite this, Docker is a prominent and well-documented technology in software development, allowing even inexperienced developers to create Docker images. The platform also collected all the results output by the algorithms to the specified results folder and processed them. Hence, we can conclude that the platform can indeed run any algorithm should it be configured correctly. It is important to note that the algorithms used were not developed specifically for the TRUST-AI platform and were the property of researchers at INESC TEC. As such, the results files had to be adapted to the TRUST-AI requirements, laid out in Section 5.4.2.

6.4 Performance Tests

Performance tests measure the amount of resources consumed by a system during the execution of a task. In terms of the performance tests done for the TRUST-AI platform, the task measured

Table 6.2: Algorithms tested in the TRUST-AI platform.

Name	**Framework**	**Language**	**Type**	**Problem**
TSP	ECJ [29]	Java	Prescriptive	Travelling salesman problem [15]
JSSP	ECJ [29]	Java	Prescriptive	Job shop scheduling problem [9]
GPG	GP-Gomea [33]	Python / C++	Predictive	Breast cancer diagnostic [27]
CP	tensorflow [44]	Python	Prescriptive	Cart pole balancing problem [6]
MD	tensorflow [44]	Python	Prescriptive	Meal delivery problem [18]

was the execution of an algorithm, which is the platform's primary focus since it was developed to run any XAI algorithm. For this effect, the results measured were in the form of time, CPU usage, and memory usage since these are directly attributed to the performance of an algorithm run.

The results were obtained with the usage of the command: `kubectl top pods`, which retrieves the CPU usage in permille (‰) and memory usage in mebibytes (MiB). For this command to be available, we need to have the kube-state-metrics[4] (KSM) Pod running in our cluster. KSM is a simple service that listens to the Kubernetes API and generates metrics about the state of the objects present in the cluster [32]. The command is called every minute, and its output, along with the current date and time, was appended to a file for later data processing.

Two different algorithms were chosen from Table 6.2 used in these tests: the JSSP algorithm, developed in Java with the ECJ framework, and the GPG algorithm, developed in Python. These algorithms were chosen because they are used to solve real-world problems and use different languages. The JSSP algorithm was tested with two different configurations, one that took a short amount of time and another that took much longer, allowing us to compare the platform overhead's effect on the algorithm execution time. On the other hand, the GP-Gomea algorithm was tested only with a short execution time configuration since the resources needed to run this algorithm for more extended periods are much higher than the ones we had available. The configurations used to obtain the results can be seen in Table 6.3.

Table 6.3: Algorithms used to obtain results and their configurations.

Algorithm	**Configuration name**	**Population Size**	**Generations**	**Seed**
JSSP	JSSP 1	20	10	-2111287466
JSSP	JSSP 2	200	50	-2111287466
GPG	GPG 1	4096	200	15682681

6.4.1 Results

In order to have a benchmark for the execution times and resource usage of the algorithms, we ran the algorithms in both the TRUST-AI platform and in a separate Kubernetes cluster as a Kubernetes Job. The latter differs from running the algorithm in the TRUST-AI platform by not having to download both the configuration and the dataset files, call the platform when it finishes, and also

[4]Found at https://github.com/kubernetes/kube-state-metrics

having its intermediate results downloaded while it is still running. This way, we can measure the full extent of the overhead introduced by the platform. The following results tables describe these methods of executing the algorithm runs as “Platform” and “Isolation”, respectively.

Each algorithm was run five times with the same configuration and seed. Their results were compiled into Tables 6.4 and 6.5 and their full extent can be seen in Appendix B. These tables show the average execution times and CPU and memory usage of the algorithm runs. The execution times are split into “Platform Time” and “Algorithm Time”, where the first is the time it took from Kubernetes Job creation to deletion, and the second is the time between algorithm start and finish as calculated by the algorithm code. This way, we can calculate the overhead introduced by both Kubernetes and the platform, which can be seen in “Overhead Time”. Each configuration was run twice, one in “Isolation” and another in “Platform”. These results were compared with each other for the same configuration, which can be seen in the “Platform” table rows as the increase in the percentage of the times registered in Table 6.4, and as the absolute value difference in Table 6.5.

Table 6.4: Times of algorithm runs in the platform with percentage comparison between the methods used for each configuration.

Configuration	**Method**	**Platform Time (s)**	**Algorithm Time (s)**	**Overhead Time (s)**
JSSP 1	Isolation	608.200	596.542	11.658
	Platform	664.598	643.531	21.068
		(+9.27%)	(+7.88%)	(+80.71%)
JSSP 2	Isolation	33405.250	32988.277	416.973
	Platform	37419.596	36810.019	609.577
		(+12.02%)	(+11.59%)	(+46.19%)
GPG 1	Isolation	149.400	143.608	5.792
	Platform	172.231	138.463	33.768
		(+15.28%)	(-3.58%)	(+382.03%)

Table 6.5: Resource usage of algorithms during algorithm runs with absolute value comparison between the methods used for each configuration.

Configuration	**Method**	**CPU ‰**	**Memory (MiB)**
JSSP 1	Isolation	1982.058±140.872	576.176±48.704
	Platform	2093.290±128.528	1666.675±180.270
		(+111.232)	(+1090.499)
JSSP 2	Isolation	1987.749±13.054	568.772±5.438
	Platform	2043.912 ± 6.999	1755.438±114.878
		(+56.163)	(+1186.666)
GPG 1	Isolation	4080.000±561.726	2675.300±301.355
	Platform	4139.600±914.969	2755.100±435.214
		(+59.600)	(+79.800)

6.4.2 Analysis

Analyzing the results presented in Section 6.4.1 we must now verify if the platform performs as expected. From a first look at the Table 6.4 it is possible to see that there is an increase of 10% in the time it takes to run the algorithm in the platform, which can be seen in the column "Platform Time", which also leads to an increase in the "Overhead Time" of the algorithm runs. This is an expected overhead of the platform since it introduces some steps to the algorithm run that are not present when the algorithm is run in Isolation. Namely, the download of the dataset and configuration, as well as the notification to the backend when the algorithm is finished, which are described in Section 5.2.1,

A slight increase in both memory and CPU consumptions, seen in Table 6.5 was also expected, not only for the same reason as for the increase in run time but also because of the intermediate result collection process that is done every five minutes. This effect is more noticeable in the JSSP algorithm runs since the run times of the GPG algorithm are shorter than the intermediate result collection cycle, meaning that there is a possibility that this process did not occur in any of the algorithm runs. In order to verify that the increase stems from this cycle, further tests must be conducted that include the number of times the data-acquisition process was executed. However, if we exclude the data obtained from the GPG 1 runs, the data gathered for the JSSP 1 and JSSP 2 runs allow us to conclude that the expected increase is about 1000 MiB of memory and 5% of the total CPU usage. This increment is in an acceptable range in terms of memory consumption since this platform is developed for cloud servers with up to 64 GB of memory.

6.5 Live Demo

The TRUST-AI platform was tested during a meeting with the TRUST-AI partners. In this meeting, a total of 24 attending people were assigned to a group, dividing the attendees into eight groups with various ranges of expertise in XAI algorithms. This was done so that each group has an Algorithm Expert, a Model Developer, and a Domain Expert, allowing for feedback collection from each platform actor. The platform was set up the same way the performance tests (described in Section 6.4) were conducted. Each group accessed the virtual machine using Chrome Remote Desktop[5], where the platform was running with the interfaces developed in [22] in a production environment. However, for simplicity and time reasons, only the JSSP algorithm was available on the platform during the demo. This way, the groups' attention was focused on using the platform and processing the results using the interface. The live demo consisted of executing three tasks: running the JSSP algorithm with the provided dataset, filtering and comparing the population results, and editing and evaluating the filtered results. The overall result of the live demo was a success, with all groups being able to complete their tasks during the assigned slot time of the meeting and without any crashes of the TRUST-AI platform.

[5]Found at https://remotedesktop.google.com/

6.5.1 Feedback

The live demo ended with filling out a feedback form designed for frontend evaluation. The results of this form can be seen in [22]. Despite the form being designed for the frontend, the last question was open, allowing additional feedback to be collected about any aspect of the platform. This question's answers were scanned through for any feedback that might apply to the backend of the platform, which is the following:

1. "Update metrics after changing formula."
2. "When we extend or edit the expression you should be able to simulate again."
3. "More information about the expression is required. How do we write the expression?"
4. "The learning process should have a progress bar (20 of 50 generations)."

6.5.2 Discussion

From the feedback received in Section 6.5.1 three different functionalities were requested to be implemented in the backend to help research/developers easily develop XAI models:

1. Update of the metrics of a formula or model output by an algorithm (feedback 1 and 2) - This feature was planned, and the requirement R11 was written for it in Section 3.1. However, due to time constraints and prioritization of the tasks, this requirement was not implemented in the platform.

2. Definition of the expressions' grammar (feedback 3) - This feature is highly specialized for XAI algorithms that output an expression, which is the case for the algorithm used in the live demo and the algorithms used for testing the platform. For this to be developed in the platform, we would have to specify the type of algorithm running as one of the available types, such as GP, decision tree, neural network, and many more. Along with this, we would need to develop a data structure that supports these types, where information such as the expression's grammar could be explicitly stored for GP algorithms. This feature is significant for the future development of the TRUST-AI platform. However, the initial focus was on having a generic platform capable of running any algorithm.

3. Progress tracking through statistics (feedback 4) - Tracking the progress of an algorithm run through statistics can be easily implemented in the platform. However, each algorithm type has different statistics. For instance, GP can use generations to track training progress, where reaching a certain number of generations terminates it. In contrast, neural networks use runs for tracking, where each run is a passage through the dataset. This would require implementing a system similar to the previous functionality requested. The difference is that this functionality requires that values usually defined in the algorithm's parameters be saved in a specific data structure and not in the JSON Schema of the algorithm, making

these statistics a requirement of the algorithm type. As such, adding this feature would make introducing an algorithm more complex, which is undesirable.

Having discussed the possible implementations and drawbacks of the said implementations, we can conclude that this feedback is helpful for the further development of the platform. As such, this feedback should guide the next steps of the implementation of the TRUST-AI platform.

Chapter 7

Conclusions

The last chapter presents our concluding remarks on the work done in this thesis in Section 7.1, and an overview of future work is presented in Section 7.2.

7.1 Final Considerations

For the TRUST-AI platform, we established four different research objectives in Section 1.2, these were: the design of a scalable infrastructure to house the TRUST-AI platform; the control of the XAI algorithms present in the platform; the abstraction of the different paradigms of AI; and management of the communication between the different components of TRUST-AI.

The first research objective tackled was the design of TRUST-AI's scalable infrastructure, as the platform cannot be developed without it. To achieve the referred scalability, we took advantage of the containerization of applications. Even though its scalability was not empirically tested in Chapter 6, it was incorporated as a design principle while also being ensured by the technologies used to develop the platform.

Once the infrastructure was designed, we moved on to implementing the platform and its communication with the Interfaces component of TRUST-AI, allowing visible progress throughout the development cycle. Once the basis of this communication was done, the second research objective was implemented by managing the XAI algorithms using a periodic process that tracks the algorithm runs and a method of controlling the end of these runs. As such, the platform has complete control of both the algorithm run status and its outputted data, described in Section 5.2.

The third research object, the abstraction of the different paradigms of AI, was done as a byproduct of the architecture design and the processing of the output of algorithms, as the framework designed for the algorithm data allows for the abstraction of the algorithm's paradigm. This objective is validated in Section 6.3, where we show that we can run algorithms of different paradigms in our platform.

The final research objective of handling communication with the other TRUST-AI components was only implemented for the Interfaces, AI Engine, and Database components, as the Cognitive Models component was not yet developed. Despite this, we still developed a communication method capable of handling these communications with a future implementation of the Cognitive Models component and any other that may be added to TRUST-AI, as discussed in Section 5.7.

In Chapter 6, a discussion of the work done in this thesis was performed, along with performance tests and the collection of user feedback obtained after a live demo of the platform. With it, we can conclude that the work implemented in this thesis does indeed achieve the established research objectives.

7.2 Future Work

In terms of future work for the TRUST-AI platform, there is still the need to implement the requirements R10 and R11 of Table 3.1, which describe the communication with the Cognitive Models and the run of specific models in the algorithm, respectively. Furthermore, the developed platform is a prototype and can be improved and optimized like any software developed. One of the ways it can be improved is by developing the features requested by the live demo users, detailed in Section 6.5.2. These features would add a great deal of specification to the TRUST-AI platform's algorithms, resulting in a trade-off in the platform's usability between Algorithm Developers and Model Developers. As such, these features need to be studied to evaluate the resulting trade-off and its impact.

Another possible path for the future work of the TRUST-AI platform is the reduction of the requirements imposed on the algorithms, whether it is the programs required by TRUST-AI or the output definition, detailed in section 5.4. As a result, the usability of TRUST-AI would increase, leading to higher adoption of the platform by Algorithm Developers. Possibly cascading into a higher adoption of TRUST-AI by Model Developers due to a higher number of algorithms available, and also Domain Experts, which follow the Model Developers.

Bibliography

[1] Martin L. Abbot and Michael T. Fisher. *The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise (2nd Edition)*. Vol. 2. Addison-Wesley Professional, 2014, pp. 1–2. ISBN: 978-0134032801. URL: `http://www.amazon.com/Art-Scalability-Architecture-Organizations-Enterprise/dp/0134032802/`.

[2] I. Ahmed, G. Jeon, and F. Piccialli. “From Artificial Intelligence to Explainable Artificial Intelligence in Industry 4.0: A Survey on What, How, and Where”. English. In: *IEEE Transactions on Industrial Informatics* 18.8 (2022). Cited By :2, pp. 5031–5042. URL: `www.scopus.com`.

[3] Ana Azevedo and Manuel Filipe Santos. “KDD, SEMMA and CRISP-DM: a parallel overview”. In: *IADIS European Conference on Data Mining 2008, Amsterdam, The Netherlands, July 24-26, 2008. Proceedings*. Ed. by Ajith Abraham. IADIS, 2008, pp. 182–185.

[4] Francisco Baeta et al. “Speed benchmarking of genetic programming frameworks”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, June 2021. DOI: `10.1145/3449639.3459335`. URL: `https://doi.org/10.1145%2F3449639.3459335`.

[5] Alejandro Barredo Arrieta et al. “Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI”. In: *Information Fusion* 58 (2020), pp. 82–115. ISSN: 1566-2535. DOI: `https://doi.org/10.1016/j.inffus.2019.12.012`. URL: `https://www.sciencedirect.com/science/article/pii/S1566253519308103`.

[6] Jason Brownlee. *THE POLE BALANCING PROBLEM A Benchmark Control Theory Problem*. `https://researchbank.swinburne.edu.au/items/62a8df69-4a2c-407f-8040-5ac533fc2787/1/`. 2005.

[7] Bernardo Carvalho, Carlos Henrique, and Carlos Mello. “Scrum agile product development method -literature review, analysis and classification”. In: *Product: Management & Development* 9 (Jan. 2011), pp. 39–49. DOI: `10.4322/pmd.2011.005`.

[8] Daswin De Silva and Damminda Alahakoon. *An Artificial Intelligence Life Cycle: From Conception to Production*. 2021. DOI: `10.48550/ARXIV.2108.13861`. URL: `https://arxiv.org/abs/2108.13861`.

[9] Cristiane Ferreira, Gonçalo Figueira, and Pedro Amorim. "Optimizing Dispatching Rules for Stochastic Job Shop Scheduling". In: *Advances in Intelligent Systems and Computing* 923 (2020), pp. 321–330. ISSN: 21945365. DOI: 10.1007/978-3-030-14347-3_31/FIGURES/3. URL: https://link.springer.com/chapter/10.1007/978-3-030-14347-3_31.

[10] Julie Gerlings, Arisa Shollo, and Ioanna Constantiou. "Reviewing the Need for Explainable Artificial Intelligence (xAI)". In: *Proceedings of the Annual Hawaii International Conference on System Sciences* 2020-January (Dec. 2020), pp. 1284–1293. ISSN: 15301605. DOI: 10.24251/hicss.2021.156. URL: https://arxiv.org/abs/2012.01007v2.

[11] Hassan B. Hassan, Saman A. Barakat, and Qusay I. Sarhan. "Survey on serverless computing". In: *Journal of Cloud Computing 2021 10:1* 10 (1 July 2021), pp. 1–29. ISSN: 2192-113X. DOI: 10.1186/S13677-021-00253-7. URL: https://link.springer.com/articles/10.1186/s13677-021-00253-7%20https://link.springer.com/article/10.1186/s13677-021-00253-7.

[12] Mir Riyanul Islam et al. "A Systematic Review of Explainable Artificial Intelligence in Terms of Different Application Domains and Tasks". In: *Applied Sciences* 12.3 (2022). ISSN: 2076-3417. DOI: 10.3390/app12031353. URL: https://www.mdpi.com/2076-3417/12/3/1353.

[13] N. Javed, F. Gobet, and P. Lane. "Simplification of genetic programs: a literature survey". English. In: *Data Mining and Knowledge Discovery* (2022). URL: www.scopus.com.

[14] S B Kotsiantis. "Decision trees: a recent overview". In: *Artif Intell Rev* 39 (Apr. 2013), pp. 261–283. DOI: 10.1007/s10462-011-9272-4.

[15] J. K. Lenstra and A. H.G. Rinnooy Kan. "Some Simple Applications of the Travelling Salesman Problem". In: *https://doi.org/10.1057/jors.1975.151* 26 (4 i 2017), pp. 717–733. ISSN: 0160-5682. DOI: 10.1057/JORS.1975.151. URL: https://www.tandfonline.com/doi/abs/10.1057/jors.1975.151.

[16] Pantelis Linardatos, Vasilis Papastefanopoulos, and Sotiris Kotsiantis. "Explainable AI: A Review of Machine Learning Interpretability Methods". In: *Entropy* 23.1 (2021). ISSN: 1099-4300. DOI: 10.3390/e23010018. URL: https://www.mdpi.com/1099-4300/23/1/18.

[17] Dang Minh et al. "Explainable artificial intelligence: a comprehensive review". In: *Artificial Intelligence Review* 55 (2021), pp. 3503–3568. DOI: 10.1007/s10462-021-10088-y. URL: https://doi.org/10.1007/s10462-021-10088-y.

[18] Damián Reyes et al. "The Meal Delivery Routing Problem". In: *Optimization Online*. 2018.

[19] M. A. Shahin. "A review of artificial intelligence applications in shallow foundations". English. In: *International Journal of Geotechnical Engineering* 9.1 (2015). Cited By :26, pp. 49–60. URL: www.scopus.com.

[20] Yan Yan Song and Ying Lu. “Decision tree methods: applications for classification and prediction”. In: *Shanghai Archives of Psychiatry* 27 (2 Apr. 2015), p. 130. ISSN: 10020829. DOI: 10.11919/J.ISSN.1002-0829.215044. URL: /pmc/articles/PMC4466856/%20/pmc/articles/PMC4466856/?report=abstract%20https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4466856/.

[21] Kashvi Taunk et al. “A Brief Review of Nearest Neighbor Algorithm for Learning and Classification”. In: *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*. 2019, pp. 1255–1260. DOI: 10.1109/ICCS45141.2019.9065747.

[22] João Varela. “User Interfaces for human-guided Explainable AI”. In: (2022).

[23] Giulia Vilone and Luca Longo. *Explainable Artificial Intelligence: a Systematic Review*. 2020. DOI: 10.48550/ARXIV.2006.00093. URL: https://arxiv.org/abs/2006.00093.

[24] Marco Virgolin, Tanja Alderliesten, and Peter A.N. Bosman. “On explaining machine learning models by evolving crucial and compact features”. In: *Swarm and Evolutionary Computation* 53 (Mar. 2020), p. 100640. DOI: 10.1016/j.swevo.2019.100640. URL: https://doi.org/10.1016%5C%2Fj.swevo.2019.100640.

[25] C. Zanni-Merk. “On the Need of an Explainable Artificial Intelligence”. In: *Information Systems Architecture and Technology: Proceedings of 40th Anniversary International Conference on Information Systems Architecture and Technology - ISAT 2019. Advances in Intelligent Systems and Computing (1050)* (2020). DOI: 10.1007/978-3-030-30440-9_1. URL: https://www.engineeringvillage.com/share/document.url?mid=inspec_5190fbdf17372a3f548M35cb10178163190&database=ins.

Web Bibliography

[26] Peter Bosman. *EA Visualizer*. http://www.cems.uwe.ac.uk/~apipe/Int%20and%20Adapt%20Sys/Revision%20material%20CD%20image/evonet.dcs.napier.ac.uk/demo7.html. Accessed: 2022-07-03.

[27] *Breast Cancer Wisconsin (Diagnostic) Data Set | Kaggle*. https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data. Accessed: 2022-07-05.

[28] *Cloud AutoML Custom Machine Learning Models | Google Cloud*. https://cloud.google.com/automl. Accessed: 2022-03-01.

[29] *ECJ*. https://cs.gmu.edu/~eclab/projects/ecj/. Accessed: 2022-07-02.

[30] *Empowering App Development for Developers | Docker*. https://www.docker.com/. Accessed: 2022-03-01.

[31] *Eureqa - Creative Machines Lab - Columbia University*. https://www.creativemachineslab.com/eureqa.html. Accessed: 2022-03-01.

[32] *GitHub - kubernetes/kube-state-metrics: Add-on agent to generate and expose cluster-level metrics*. https://github.com/kubernetes/kube-state-metrics. Accessed: 2022-03-01.

[33] *GitHub - marcovirgolin/GP-GOMEA: Genetic Programming version of GOMEA. Also includes standard tree-based GP, and Semantic Backpropagation-based GP*. https://github.com/marcovirgolin/GP-GOMEA. Accessed: 2022-07-02.

[34] *H2O.ai | AI Cloud Platform*. https://live-h2oai.pantheonsite.io/. Accessed: 2022-03-01.

[35] *HeuristicLab*. https://dev.heuristiclab.com/trac.fcgi/. Accessed: 2022-03-01.

[36] *JavaScript | MDN*. https://developer.mozilla.org/en-US/docs/Web/JavaScript. Accessed: 2022-07-03.

[37] *JetStream - NATS Docs*. https://docs.nats.io/nats-concepts/jetstream. Accessed: 2022-07-02.

[38] *Kubeflow*. https://www.kubeflow.org/. Accessed: 2022-03-01.

[39] *Kubernetes*. https://kubernetes.io/. Accessed: 2022-03-01.

[40] *Microservice Architecture pattern*. `https://microservices.io/patterns/microservices.html`. Accessed: 2022-03-01.

[41] *NATS.io – Cloud Native, Open Source, High-performance Messaging*. `https://nats.io/`. Accessed: 2022-03-01.

[42] *Serverless: Develop & Monitor Apps On AWS Lambda*. `https://www.serverless.com/`. Accessed: 2022-03-01.

[43] *Symbolic regression software - TuringBot*. `https://turingbotsoftware.com/`. Accessed: 2022-03-01.

[44] *TensorFlow*. `https://www.tensorflow.org/`. Accessed: 2022-07-02.

[45] *TRUST-AI | Home*. `http://trustai.eu/`. Accessed: 2022-01-20.

[46] *TypeScript: JavaScript With Syntax For Types*. `https://www.typescriptlang.org/`. Accessed: 2022-07-03.

Appendix A

Server Specifications

The server used for the platform evaluation in Chapter 6 has the following specifications:

CPU - AMD Ryzen Threadripper 3990WX, 64 cores, 2700 Mhz and 128 logical processors

Memory - 256 Gigabytes, 3200 Mhz

Operating System - Windows 11

However, in the evaluation process itself, via the VMWare software, we only allocated to the virtual machine running the TRUST-AI platform 16 processors and 24 GB of RAM in an Ubuntu MATE operating system.

Appendix B

Performance Test Results

Table B.1: Time results of algorithm runs in performance tests.

Configuration	**Method**	**Run**	**Platform Time (s)**	**Algorithm Time (s)**	**Overhead Time (s)**
JSSP 1	Platform	1	658.300	649.456	8.844
JSSP 1	Platform	2	669.844	647.137	22.707
JSSP 1	Platform	3	654.632	633.454	21.178
JSSP 1	Platform	4	659.790	635.573	24.217
JSSP 1	Platform	5	680.426	652.033	28.393
JSSP 1	Isolation	1	594.000	583.423	10.577
JSSP 1	Isolation	2	585.000	574.400	10.600
JSSP 1	Isolation	3	576.000	565.775	10.225
JSSP 1	Isolation	4	586.000	573.943	12.057
JSSP 1	Isolation	5	700.000	685.167	14.833
JSSP 2	Platform	1	37434.330	36726.754	707.576
JSSP 2	Platform	2	37603.709	37129.835	473.874
JSSP 2	Platform	3	37815.828	37405.899	409.929
JSSP 2	Platform	4	36847.776	36114.598	733.178
JSSP 2	Platform	5	37396.335	36673.007	723.328
JSSP 2	Isolation	1	31855.000	31237.135	617.865
JSSP 2	Isolation	2	33798.000	33793.137	4.863
JSSP 2	Isolation	3	33348.000	32761.447	586.553
JSSP 2	Isolation	4	34620.000	34161.390	458.610
JSSP 2	Isolation	5	34916.000	34329.701	586.299
GP-Gomea 1	Platform	1	169.694	137.527	32.167

Continued on next page

Table B.1 – continued from previous page

Configuration	**Method**	**Run**	**Platform Time (s)**	**Algorithm Time (s)**	**Overhead Time (s)**
GP-Gomea 1	Platform	2	169.996	137.452	32.544
GP-Gomea 1	Platform	3	177.259	138.604	38.655
GP-Gomea 1	Platform	4	170.377	138.594	31.783
GP-Gomea 1	Platform	5	173.829	140.137	33.692
GP-Gomea 1	Isolation	1	150.000	144.417	5.583
GP-Gomea 1	Isolation	2	149.000	143.060	5.940
GP-Gomea 1	Isolation	3	149.000	143.461	5.539
GP-Gomea 1	Isolation	4	150.000	144.543	5.457
GP-Gomea 1	Isolation	5	149.000	142.560	6.440

Table B.2: Resource usage of algorithms in performance tests.

Configuration	**Method**	**Run**	**Measu-rements**	**CPU (m)**	**Memory (MiB)**
JSSP 1	Platform	1	9	2087.667 ± 132.320	1610.444 ± 744.879
JSSP 1	Platform	2	10	2001.800 ± 355.335	1682.900 ± 512.414
JSSP 1	Platform	3	8	2034.375 ± 308.547	1691.000 ± 694.810
JSSP 1	Platform	4	9	2026.333 ± 352.031	1423.667 ± 438.334
JSSP 1	Platform	5	11	2316.273 ± 259.788	1925.364 ± 665.820
JSSP 1	Isolation	1	9	1920.778 ± 497.741	580.444 ± 183.252
JSSP 1	Isolation	2	10	1813.800 ± 308.893	510.600 ± 85.250
JSSP 1	Isolation	3	8	2151.125 ± 210.292	618.625 ± 150.530
JSSP 1	Isolation	4	9	2104.222 ± 166.335	545.667 ± 21.662
JSSP 1	Isolation	5	11	1920.364 ± 342.609	625.545 ± 152.088
JSSP 2	Platform	1	623	2055.650 ± 322.790	1948.332 ± 933.760
JSSP 2	Platform	2	626	2041.000 ± 320.833	1772.919 ± 730.944
JSSP 2	Platform	3	629	2037.092 ± 309.836	1694.154 ± 660.391
JSSP 2	Platform	4	614	2042.138 ± 309.455	1666.671 ± 573.267
JSSP 2	Platform	5	622	2043.680 ± 329.346	1695.116 ± 649.217
JSSP 2	Isolation	1	530	1984.515 ± 304.101	567.257 ± 101.886
JSSP 2	Isolation	2	563	1982.941 ± 308.093	561.811 ± 99.528
JSSP 2	Isolation	3	555	1976.849 ± 300.253	572.000 ± 112.150
JSSP 2	Isolation	4	577	2006.692 ± 304.081	574.021 ± 92.840
JSSP 2	Isolation	5	581	1984.167 ± 312.901	574.578 ± 97.334
GP-Gomea 1	Platform	1	2	3747.000 ± 1306.733	2734.500 ± 620.133

Continued on next page

Table B.2 – continued from previous page

Configuration	**Method**	**Run**	**Measu-rements**	**CPU (m)**	**Memory (MiB)**
GP-Gomea 1	Platform	2	2	3373.500 ± 1395.122	2987.000 ± 222.032
GP-Gomea 1	Platform	3	2	5239.500 ± 1180.161	3253.500 ± 372.645
GP-Gomea 1	Platform	4	2	5004.500 ± 579.120	2719.000 ± 653.367
GP-Gomea 1	Platform	5	2	3333.500 ± 1711.906	2081.500 ± 1477.146
GP-Gomea 1	Isolation	1	2	3569.000 ± 1357.645	2668.000 ± 595.384
GP-Gomea 1	Isolation	2	2	3488.500 ± 2199.809	2549.500 ± 740.341
GP-Gomea 1	Isolation	3	2	4107.500 ± 2083.844	2632.500 ± 697.914
GP-Gomea 1	Isolation	4	2	4808.000 ± 1511.794	2357.000 ± 1327.947
GP-Gomea 1	Isolation	5	2	4427.000 ± 1306.733	3169.500 ± 358.503