A Model-Driven Engineering Approach for Monitoring ML Model Performance

Panagiotis Kourouklidis

Doctor of Philosophy

University of York
Department of Computer Science

October 2023

Abstract

With a rising number of enterprises adopting machine learning (ML) in their operations, the issue of ML monitoring to ensure robustness has become increasingly relevant. Unfortunately, implementing ML monitoring systems has proven challenging partly because it requires cross-discipline collaboration between data scientists and software engineers. This thesis hypothesises that a solution centred around model-driven engineering (MDE) comprising a domain-specific language and an accompanying execution environment can address many of the challenges associated with ML monitoring. To evaluate the validity of this hypothesis, such a solution was designed at the architectural level and implemented. The solution's design offers portability, extensibility and separation of concerns between data scientists and software engineers. This is validated through empirical studies involving professional data scientists. In addition, three case studies with third-party ML models have been developed to further evaluate the solution's validity.

Acknowledgements

I would like to express my sincere gratitude to all those who have helped me during my doctorate studies. I would first like to thank my wife, Dobby Võ, whose emotional support during the challenging moments was instrumental to the successful completion of my studies. I would also like to thank my supervisors, Prof. Dimitris Kolovos, Dr. Joost Noppen and Dr. Nicholas Matragkas, for giving me a great opportunity. Special thanks go to Dimitris, whose quick and detailed comments were immensely helpful, especially during the final phase of writing this thesis.

I would like to express my warmest gratitude to my parents, Leonidas Kourouklidis and Katerina Tsaggaraki as well as my brother Giannis Kourouklidis, whose unconditional love has been a source of strength throughout my life.

I would also like to thank my internal examiner, Prof. Radu Calinescu, and external examiner, Prof. Javier Luis Cánovas Izquierdo, whose detailed comments have helped me improve this thesis.

I am indebted to Prof. Nikolaos Mitianoudis and Prof. Christodoulos Chamzas, whose flattering recommendation letters undoubtedly played a big role in my selection to participate in the Lowcomote project.

Finally, this work would not have been possible without the financial support of the European Commission via the Lowcomote project, which received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 813884.

Author Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for a degree or other qualification at this University or elsewhere. Parts of this work have been previously published by the author at a MODELs 2020 workshop [1], at a MODELS 2021 workshop [2] and has been accepted for publication at SAM 2023 [3] .

Dedication

Στην μνήμη του θείου Πόλυ, του παππού Γιάννη, της γιαγιάς Αναστασίας και του παππού Παναγιώτη

Contents

List of Figures

Chapter 1

Introduction

Models, as in abstract representations of real-world phenomena or systems, are widely used in science and engineering. A distinction can be made between descriptive and prescriptive models [5]. Most models produced in the natural sciences are of the descriptive kind. As the name suggests, the purpose of a descriptive model is to describe phenomena or systems that exist in the real world, often for the purpose of explaining the inner workings of the subject or predicting its future state. A weather model produced by a meteorologist and utilised to make weather forecasts is a prime example of a descriptive model.

On the other hand, the subject of a prescriptive model does not yet exist. The model, once produced, will be used to construct the modelled subject and thus contains all the information needed to do so. Examples of prescriptive models are the blueprints that are used in building construction.

This thesis deals with models of both kinds. On the one hand, machine learning models can be classified as descriptive, as they seek to model real-world phenomena typically for the purpose of making predictions. Models in the context of model-driven engineering, on the other hand, are typically prescriptive, and this is also the case for the ones produced in the context of this thesis. These models will be utilised to prescribe the behaviour of software systems.

This chapter presents a brief overview of the challenges that motivated the work presented in

this thesis. It then outlines the research hypothesis and summarises the thesis's results and main contributions. Finally, it provides an overview of the organisation of the thesis and a summary of the remaining chapters.

1.1 Motivation

Machine learning (ML) models have been transformative for various sectors, driving them to become more data-centric. However, as these ML models play a more decisive role, ensuring consistency and bias-free operation is crucial. While a model might perform well during its initial training, the ever-evolving nature of the real world means its effectiveness might not remain consistent. Hence, continuous monitoring of ML models is essential, especially in business contexts where consistent performance is vital.

However, implementing ML monitoring is difficult. It demands the combined efforts of software engineers and data scientists, a cross-disciplinary collaboration that poses unique challenges [6]. To resolve these issues, it is essential to facilitate a clear separation of concerns between the practitioners of the two fields.

On the other hand, in the field of model-driven engineering, there is an established practice of developing Domain Specific Languages (DSLs). As these languages are narrowly focused on a single domain, they can cater to the needs of the domain's experts more extensively. This approach could be followed to alleviate the challenges of the ML monitoring domain.

1.2 Hypothesis and Objectives

In this context, the hypothesis of this thesis is stated as follows:

> Commonly used supervised machine learning models exhibit diversity in their target tasks, input and output modalities, and implementation technologies. Despite that, it is hypothesised that a unifying representation of the workflows for monitoring

against and mitigating potential performance degradation can be formulated as a declarative domain-specific language (DSL). Additionally, this DSL can be agnostic towards the underlying ML platform so that experts in statistical modelling who might lack an extensive software engineering skillset can use it to specify and deploy such workflows unassisted. Finally, the execution environment for workflows specified in this DSL can be designed to interface with a wide variety of ML platform components to facilitate portability and extensibility.

Based on this research hypothesis, the objectives of the thesis are formulated as follows:

1. The design of a declarative domain-specific language for specifying ML monitoring workflows and implementing an accompanying execution environment.
   (a) The solution must enable data scientists to deploy ML monitoring workflows independently, without direct assistance from software engineers. In other words, it must enable the *separation of concerns* between data science and software engineering disciplines.
   (b) The overall system must offer *extensibility* by integrating dataset-shift-detection algorithms created by data scientists written in the programming language of their choice. Additionally, it must offer the possibility for software engineers to add software components to support the execution of monitoring workflows across different implementation technologies.
   (c) The solution must be designed for *portability*. This means that it must be agnostic towards the underlying computing infrastructure and not be strongly coupled to any proprietary technologies limited to specific vendors.
2. Evaluate the solution from multiple angles. Namely, usability, domain coverage and ability to lower technical barriers for data scientists.
   (a) To evaluate usability, design a laboratory environment to measure whether data scientists can complete, unassisted, the specification of monitoring workflows using the developed solution.

(b) To evaluate domain coverage, carry out several case studies that include ML models with different target tasks, input and output modalities and implementation technologies. Additionally, the expert opinion of data scientists can be called upon to judge domain coverage and increase confidence.

(c) To evaluate the extent to which technical barriers have been lowered for data scientists, ask data scientists to judge the potential of the proposed solution to aid them in the implementation of ML monitoring workflows.

1.3 Research Results

The main result of this thesis is the design of a distributed architecture of loosely coupled components for a solution that targets the ML monitoring workflow domain. The solution is centred around an MDE layer that provides a DSL that data scientists can utilise to specify their desired monitoring workflows without delving into technical details.

The design and implementation of this DSL and its accompanying execution environment covers the first objective of this thesis with its three sub-objectives. Specifically, these sub-objectives are covered in the following places within the thesis:

- Chapter 3 explains the technical architecture of the solution. Furthermore, sections 3.4 and 3.5 delve into the various layers that comprise the architecture and how the design decisions taken enable separation of concerns, extensibility and portability, thus addressing all three sub-objectives.

- Chapter 4 introduces the DSL developed in the context of this thesis. The chapter shows how the declarative nature of the DSL enables data scientists to independently define and deploy monitoring workflows, thus addressing sub-objective 1a.

- Chapter 5 covers the technical implementation of the solution. It shows how the choice of suitable implementation technologies streamlines the process of deploying the solution atop different computing infrastructure thus addressing sub-objective 1c. Additionally, it

is shown how additional components that interface with the solution's main components can be implemented, thus addressing sub-objective 1b.

To satisfy the second objective, the research hypothesis has been validated by conducting three empirical studies that showcase the solution's usability, domain coverage and ability to lower the technical barriers for data scientists and enable them to deploy ML monitoring workflows more efficiently. Specifically, each of the three sub-objectives are covered in the following places within the thesis:

- Section 6.1, presents the results of a judgement study which calls upon experts to evaluate the solution's domain coverage and technical barrier lowering potential, thus addressing sub-objectives 2b and 2c.
- Section 6.2, presents the results of a laboratory experiment which seeks to evaluate the solutions usability and technical barrier lowering potential, thus addressing sub-objectives 2a and 2c.
- Section 6.3, presents the results of three experimental simulations, conducted to evaluate the solution's domain coverage, thus addressing sub-objective 2b.

1.4 Thesis Structure

Chapter 2 presents a review of relevant background material. The chapter covers the theoretical foundations of machine learning and introduces the concept of dataset shift along with its causes and techniques that can be applied to mitigate its adverse effects. In addition, the chapter presents an overview of how machine learning is applied in Industry. Finally, the chapter presents relevant background material for the field of model-driven engineering.

Chapter 3 presents a high-level overview of Panoptes, the proposed solution for facilitating the deployment of ML monitoring workflows. The chapter begins by providing the context in which the solution was developed and an illustrative example used throughout the thesis to

present the various aspects of Panoptes. Subsequently, a high-level overview of the solution's architecture and the roles and responsibilities of the different stakeholders is given. Finally, the chapter presents Panoptes' main design decisions and how they contribute towards the desired attributes of portability, extensibility and separation of concerns.

Chapter 4 presents Panoptes Description Language (PDL), the DSL developed in the context of Panoptes to allow data scientists to specify ML monitoring workflows at a high level of abstraction. The chapter starts with a description the core classes of the DSL's metamodel. Then the classes of the metamodel related to dataset shift and the ones related to scheduling are introduced. Finally, the model validation features of the DSL that can help data scientists construct correct PDL models are presented.

Chapter 5 covers the technical aspects of Panoptes that are implementation-specific and introduces Panoptes' reference implementation, built to validate the approach. The chapter covers the open-source technologies used as a base layer upon which the reference implementation was built. Additionally, the chapter introduces the format that Panoptes' components use to serialise the events they produce. The rest of the sections cover the technical details of Panoptes' major components as well as the process that a project team would need to follow to adopt the solution.

Chapter 6 presents three empirical studies conducted to evaluate the work and validate the research hypothesis. The first study was conducted in the form of structured interviews with expert data scientists who were asked to judge the proposed approach in principle. In the second study, participants were asked to use Panoptes in the context of a simulation system in order to evaluate its usability. After this hands-on experience with Panoptes, participants were again asked to evaluate it. In the third study, third-party ML models of different characteristics were used with Panoptes to evaluate its domain coverage. The chapter concludes with a discussion about potential threats to validity posed by the chosen evaluation methodology.

Chapter 7 concludes the thesis by summarising its main contributions, discussing its limitations and suggesting future work that can potentially improve the proposed solution's suitability for enterprise users.

Finally, Appendix A provides the complete Xtext grammar of PDL's textual syntax, and Appendix B provides information regarding the research ethics process that was followed for the empirical studies that involved external participants.

Chapter 2

Background

This chapters reviews the background material in the field of machine learning and model-driven engineering relevant to this thesis. Regarding machine learning, Section 2.1 provides definitions and a brief overview of the theoretical foundations. Section 2.2 analyses how machine learning models can fail in constantly-changing environments such as those encountered in the real world. Section 2.3 describes how machine learning is applied in industry. Regarding model-driven engineering, Section 2.4 covers some of the theoretical and practical aspects of the field.

2.1 Machine Learning

Since Alan Turing posed the question, "Can machines Think?" in 1950 [7], a large body of work has been devoted to artificial intelligence research. Nowadays, machine learning, which can be informally described as "learning from examples", is perceived as the most promising approach to artificial intelligence, but this has not always been the case. Based on logic rules and automated deductive procedures, Symbolic AI was the preferred approach until the 1990s [8]. In hindsight, this makes intuitive sense since, to produce results competitive with symbolic AI, machine learning requires computing resources and data quantities, which until recently were prohibitive.

The following are some of the historical milestones in the development of machine learning.

- 1962: F. Rosenblatt proposes the perceptron as the mathematical basis of a learning machine [9]. He based the perceptron on the McCulloch-Pitts model of a biological neuron [10].

- 1974-1980: First "AI winter". Due to the failure of early AI systems to live up to expectations, government funding is significantly reduced. [11]

- 1986: Backpropagation is discovered, which allows efficient training of larger neural networks [12].

- late 1980s - late 1990s: Second "AI winter" [13, 11].

- 2011: IBM's Watson beats human players in the popular American game show "Jeopardy!" [14].

- 2012: Alex-net, a deep convolutional neural network, achieves 15.36% top-five error in the ImageNet Large Scale Visual Recognition Challenge [15].

- 2015: Deepmind's AlphaGo beats Go champion Lee Se-dol [16].

- 2017: Researchers from Google introduce the transformer architecture and apply it to the natural language processing domain. The transformer proves to be a highly influential development and is consequently widely adopted [17].

2.1.1 Machine Learning Scenarios

Mohri [18] broadly defines machine learning as "computational methods using experience to improve performance or to make accurate predictions". While all scenarios in which machine learning can be applied adhere to this definition, there is some variation to the techniques based on the specifics of the scenario. Mohri specifies eight such scenarios, which include supervised learning, unsupervised learning, semi-supervised learning, transducive inference, on-line learning, reinforcement learning and active learning.

According to industry reports [19], of these machine learning scenarios, supervised learning is the one most commonly applied in practice and is also backed by rigorous mathematical theory [20]. For these reasons, this thesis specifically focuses on this type of machine learning.

2.1.2 Supervised Learning

In the supervised learning setting, one wants to predict the value of a target variable (e.g. whether a given email is spam) from the value of an observed variable (e.g. the body of the said email). We denote by X the set of possible values for the observed variable and Y the set of possible values for the target variable. Given a set of labelled examples (paired elements of X, Y), the goal is to extract a mapping from X to Y, which can be used to predict labels Y for all unseen instances of X [18]. Hastie et al. [21] further distinguish supervised learning tasks based on the target variable type. When the target variable is categorical (i.e. the set of possible values is finite), we have the classification task. On the other hand, for quantitative target variables, the task is called regression.

From a theoretical viewpoint, statistical learning theory [20] studies various aspects of the learning problem, such as the rate of convergence, consistency and generalization of the learning processes. In this theoretical framework, the learning problem is formulated as follows:

- There is a fixed but unknown probability distribution $P(x)$ from which random samples are drawn.
- For a number of samples, values y can be obtained for each value x. The samples follow the conditional distribution $P(y|x)$. This is typically referred to as the training set.
- There is a set of functions $f(x, \alpha),\ \alpha \in \Lambda$, where Λ is a set of parameters. The learning process can search this space of functions for potential solutions.

The goal of the learning process, then, is to select a function $f(x, \alpha_0)$ from the set of possible functions such that its outputs approximate the values y well. Naturally, the question “What

is a good approximation" arises. For this, the notion of the loss function is introduced. A loss value $L(y, f(x, \alpha))$ measures how close a predicted value of the target variable is to the ground truth y. Given the notion of loss, the learning process is then distilled down to selecting function $f(x, \alpha_0)$, which results in the lowest expected loss over the sample distribution. This expected loss is referred to as risk and is formally defined as:

$$R(a) = \int L(y, f(x, \alpha)) dP(x, y)$$

In most cases, the joint probability distribution function $P(x, y)$ is unknown. In such a case, the function $f(x, \alpha_0)$ that minimises the risk is approximated by the function $f(x, \alpha_l)$ that minimises the empirical risk, which is formally defined by the following with $(x_i, y_i), 1 \leq i \leq N$ being the training set samples:

$$R_{emp}(\alpha) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i, \alpha))$$

2.2 Dataset Shift

As mentioned in the previous section, the standard assumption in supervised learning is that the training set is sampled from the same probability distribution as future unlabelled samples for which the value of the target variable needs to be predicted. On the other hand, due to the dynamic and ever-changing nature of the world, the assumption of a static probability distribution often does not hold with potentially detrimental effects on the accuracy of trained ML models. This scenario has been studied in the literature under numerous terms, such as concept drift/shift [22, 23], covariate/sampling shift [24, 23, 25] and prior probability shift [23, 25].

In recent years, the more general term "dataset shift" has been introduced in [26] and further standardised in [27] and [28]. In this thesis, the terminology of [28] is adopted as the more

general one, which defines dataset shift as follows:

> Dataset shift appears when training and test joint distributions are different. That is, when $P_{tr}(y, x) \neq P_{tst}(y, x)$.

The phrase "test joint distribution" refers to the joint probability distribution of the test set. The term test set is primarily used in the machine learning domain to describe a set of samples not utilised during training and used to evaluate the performance of the trained model. This section examines the causes of dataset shift and how its various types affect the joint probability distribution of observed and target variables.

2.2.1 Causes of Dataset Shift

Storkey's work [25] explores the root causes of dataset shift by leveraging causal probabilistic models. These models are visually represented as graphs, where nodes symbolise random variables and directed edges signify a causal relationship between these variables.

Figure 2.1 provides a visual representation of two such causal graphs. The left subfigure of Figure 2.1 demonstrates a scenario where random variable X causally influences random variable Y. To illustrate this, Storkey provides an example where X symbolises a person's smoking habits, while Y denotes the likelihood of that individual developing lung cancer in their lifetime. The graph indicates that smoking habits (X) are a causal factor for lung cancer (Y), but the reverse is not true. This is intuitive, as the onset of lung cancer cannot retroactively affect one's past smoking habits. Such a causal relationship can lead to covariate shift, a form of dataset shift. Covariate shift arises when there is a change in the distribution of X due to an altered context, such as a government-imposed smoking ban in enclosed spaces. However, the probability of developing lung cancer remains consistent for individuals with the same smoking habits across different contexts. The relationship between the context and variable X is depicted in the right subfigure of Figure 2.1.

Conversely, the left subfigure of Figure 2.2 portrays a situation where Y is the causal factor for

(a) Initial Scenario (b) Covariate Shift

Figure 2.1: Covariate Shift

(a) Initial Scenario (b) Prior Probability Shift

Figure 2.2: Prior Probability Shift

X. Fawcett and Flach provide an illustrative example: the onset of specific medical symptoms (X) in individuals is caused by the presence of an infectious disease (Y). During a pandemic, when a particular disease becomes more prevalent, the associated symptoms will also manifest more frequently. A change in the distribution of Y leads to prior probability shift, another form of dataset shift. The relationship between the context and variable Y is visually represented in the right subfigure of Figure 2.2.

Building upon Storkey's work, Kull and Flach [27] propose a way to systematically model the causes of the dataset shift. They claim that their systematisation is complete regarding dataset shift causes but leave the formal proof of this claim as future work.

2.2.2 Effects of Dataset Shift on the Joint Probability Distribution

Torres et al. [28] introduce an alternative way to classify the different types of dataset shift based on the effects on the joint probability distribution. A relatively simple causal model is used based on the work of Fawcett and Flach [29] where either X causes Y or Y causes X. Based on that, the following dataset shift types are introduced:

- Covariate Shift, which is defined as the case where $P_{tr}(x) \neq P_{tst}(x)$ but $P_{tr}(y|x) = P_{tst}(y|x)$. It only appears in scenarios where X causes Y.

- Prior probability shift, which is defined as the case where $P_{tr}(y) \neq P_{tst}(y)$ but $P_{tr}(x|y) = P_{tst}(x|y)$. It only appears in scenarios where Y causes X.

- Concept shift, which is defined as $P_{tr}(y|x) \neq P_{tst}(y|x)$ and $P_{tr}(x) = P_{tst}(x)$ in scenarios where X causes Y. On the other hand, it is defined as $P_{tr}(x|y) \neq P_{tst}(x|y)$ and $P_{tr}(y) = P_{tst}(y)$ in scenarios where Y causes X.

- Dataset shift which does not fit any of the above cases is denoted "other type of dataset shift". It is defined either as $P_{tr}(y|x) \neq P_{tst}(y|x)$ and $P_{tr}(x) \neq P_{tst}(x)$ for scenarios where X causes Y or $P_{tr}(y|x) \neq P_{tst}(y|x)$ and $P_{tr}(y) \neq P_{tst}(y)$ for scenarios where Y causes X.

2.2.3 Techniques for Dataset Shift Mitigation

In the interest of robustness in the context of production-grade ML systems, practitioner-focused resources recommend the periodic execution of dataset shift detection procedures and the subsequent adaptation as needed [30, 31]. The relevant academic literature also supports the validity of this approach. Specifically, Lu et al. [32] conduct an extensive literature review of the dataset shift field and find that an effective workflow for addressing dataset shift consists of detection and adaptation steps.

Concretely put, an ML monitoring workflow consists of the following phases:

Serving phase: In this phase, an ML model has either been recently adapted to a prior dataset shift or deployed for the first time. The system is thus waiting to serve enough predictions to warrant the execution of a dataset shift detection procedure.

Detection Phase: Once enough new data points are available, a dataset shift detection algorithm is executed.

Adaptation Phase: In case the result of the algorithm execution indicates the presence of dataset shift, an action is taken to adapt the ML model to the new situation.

For the rest of this thesis, the recurring application of the three phases described above will be referred to as a monitoring workflow.

In addition to the above literature review, the following more recent works also suggest techniques for dataset shift mitigation:

- Zenisek et al. [33] describe an interesting application of dataset shift in the context of predictive maintenance of industrial machinery. Their method relies on training a regression model that predicts future states of an industrial machine. The ML model is trained on data from the machine in a healthy state. When the predictions of the ML model no longer agree with the ground truth data gathered from sensors attached to the machine (ie. dataset shift is present), this indicates that the machine is no longer in a healthy state and is in need of maintenance. What is unusual with the approach proposed in this work is that the detection of dataset shift itself is the end goal as the ML model's predictions are not useful by themselves.

- Ackerman, Raz et al. [34] propose a dataset shift detection algorithm that relies solely on classifier confidence without the need for ground truth labels. Their method assesses whether the classifier's confidence levels in production deviate significantly from those during the model's training phase. The approach employs classical statistical tests to measure this deviation, thereby determining the presence of dataset shift.

- Soin et al. [35] present an interesting application of dataset shift detection in the domain of medical imaging. The authors introduce a novel approach which once again does not require ground truth labels and test it on two open medical imaging datasets. The method relies on constructing a unified multi-model metric for each medical image based on it's metadata as well as the imaging data. For the imaging data, the method utilises variational autoencoders (VAEs) to reduce images to a latent representation. Afterwards, statistical tests like the Kolmogorov-Smirnov and chi-square tests are applied to detect

differences in the distributions of the unified metric between the reference dataset and recent data.

- Ackerman, Farchi et al. [36] propose a general methodology that has gives fewer false positives compared with a repeated application of two-sample statistical testing methods such as the Kolmogorov-Smirnov test. The authors claim that the application of the Change Point Model technique is superior for dataset shift detection as it does not look at sequentially arriving sets of data in isolation but considers the whole sequence. The methodology is empirically validated using modified MNIST datasets, where specific classes are omitted during training to simulate dataset shift during deployment. This setup tests the model's ability to detect new, unseen classes as well as changes in class distribution. The results demonstrate the robustness of the proposed method under various scenarios, including gradual and sudden drifts.

- Mirza et al. [37] present two approached to mitigate dataset shift in the domain of image segmentation. The first approach leverages Image Quality Assessment (IQA) metrics to select high-quality images for training. It aims to prevent the inclusion of poor-quality data which could degrade model performance. The second approach uses feature vectors extracted from the existing model to guide the selection and use of new data for model retraining. This approach ensures that new data aligns with what the model has previously learned, potentially increasing the robustness of the model. The two approaches are evaluated based on three open image segmentation datasets with good results.

Overall, each of the above papers proposes techniques that detect some kind of dataset shift. While some of the techniques are general and some target a specific application domain, all of are consistent with the three-phase general framework of serving, detection and adaptation. This provides additional confidence that the framework remains valid some years after its introduction.

2.3 Machine Learning in Industry

In recent years, there has been a lot of interest in leveraging machine learning approaches for commercial purposes. According to Delloite's latest state of AI report [38], 94% of business leaders agree that AI/ML is critical to success over the next five years.

This section covers the various aspects related to developing production-grade ML systems. Subsection 2.3.1 covers the stakeholders involved in the various stages of an ML project. Subsection 2.3.2 presents the typical workflow data scientists follow to develop ML models. Subsection 2.3.3 covers the platforms that organisations need to put in place to support ML workflows and presents some of the relevant commercial services. Lastly, subsection 2.3.4 presents some of the challenges organisations face when attempting to develop ML projects.

2.3.1 ML Project Stakeholders

As with traditional software projects, an ML project progresses through various stages from its inception until its eventual decommission. Throughout these stages, different stakeholder groups are involved in the project's development and are invested in and affected by its outcome. At the highest level, stakeholders can be put in two broad categories: technical and non-technical [30].

From a business point of view, the stakeholders of an ML project can include:

Business Executives/Leadership: They set the strategic direction and provide resources for the project. They are concerned about financial and commercial aspects such as Return on investment (ROI), alignment with business goals, and gaining a competitive advantage in the marketplace.

Product Managers: They define product requirements, prioritise features, and ensure the ML solution meets user needs. They are concerned with product-market fit, user adoption, and feedback.

From a technical point of view, two groups of contributors can be distinguished: data scientists and software engineers. These two groups work in collaboration in order to build an ML-enabled system successfully [6].

Firstly, there are *data scientists*. This term describes people who use statistical analysis and machine learning techniques to extract insights from large datasets. They work on data exploration, feature engineering, model selection, and evaluation. They may also be involved in data visualisation and storytelling to communicate findings effectively.

The second group are *software engineers*. This group contains many different types of contributors who specialise in different parts of an ML project. Amongst them, the following software engineering specialities are especially relevant:

Data Engineers: They are responsible for designing, building, and maintaining the infrastructure required to support machine learning projects. They develop data pipelines, manage databases, and ensure the availability and reliability of data for analysis and model training.

Infrastructure/Cloud Engineers: They manage and optimise the cloud resources or on-premises infrastructure where the ML models are trained and deployed.

DevOps Engineers: They focus on the deployment, scaling, monitoring, and continuous integration/continuous deployment (CI/CD) of the ML models and associated software. They ensure that models are deployed in a scalable and maintainable manner.

Application Engineers: They develop user interfaces and experiences that allow users to interact with ML models, visualise results, or input data.

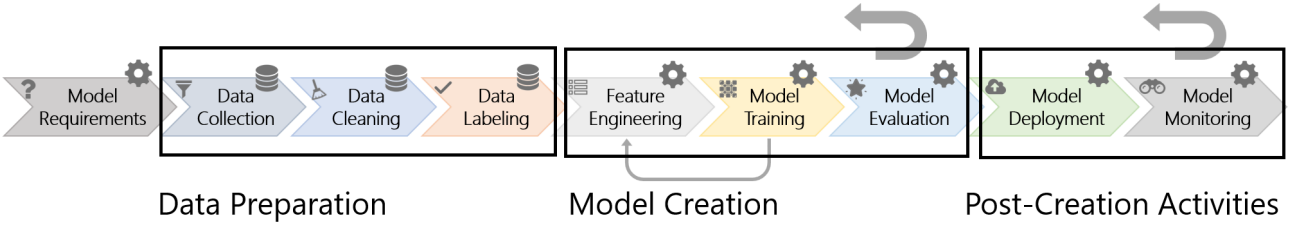


Figure 2.3: Machine Learning workflow (adapted from[4])

2.3.2 ML Workflow

Once an ML project is approved, the team tasked with the implementation needs to follow specific steps. Microsoft reports a nine-stage workflow presented by Amershi et al. [4] that they claim to be similar to other commonly followed data science workflows such as TDSP, KDD and CRISP-DM. Figure 2.3 shows the steps of the workflow grouped into three categories: data preparation, model creation and post-creation activities.

Data Preparation

As ML systems rely on data to be effective, a large portion of the machine learning workflow is devoted to data-related activities. According to an internal survey conducted by Microsoft, the top-ranked challenge in the field of machine learning, as perceived by employees working in it, is data availability, collection, cleaning, and management [4].

Due to the importance of data, organisations that integrate machine learning in their products have invested substantial engineering resources in developing systems that can validate incoming data, as discussed in [39, 40, 41]. In these publications, the authors present systems enabling users to specify the expected data properties to be received. Subsequently, the system automatically checks that the incoming data meets the specifications. The goal is to keep the quality of incoming data consistent and alert the engineers of any anomalies. By ensuring that all of the data are of high quality, they can then safely be used for training new models or fed into existing models for inference. In that regard, one could draw a parallel to software testing. Similar to how developers of traditional software products test new code to ensure it does not introduce bugs in their code base, developers of machine learning systems should test

new incoming data to ensure that they are consistent with what the system expects to receive.

Model Creation

The second group of activities in the machine learning workflow has to do with the creation of the model artefact. These are perhaps the activities mostly perceived to be the core of machine learning but are usually just a small part of the overall system in terms of code volume [42]. The model created at this stage results from an iterative process by which different aspects of the raw data, algorithms and input parameters for these algorithms are experimented with to determine which combination delivers the best performance. Keeping in mind that this experimentation might take place over several days and be performed collaboratively by several data scientists, it becomes evident that there is a need for a system that can keep track of the performance metrics of every combination of factors that was attempted.

Such systems have indeed been designed and used by teams that train machine learning models. The systems encountered in the literature [43, 44] follow a similar approach. Every time a machine learning model is trained, the system stores a variety of metadata in a database for future reference. The metadata stored can include the dataset used for training and evaluating the model, the performance metrics achieved by the model when evaluated, the input parameters used for the training of the model and also custom fields that the developer wants to associate with that particular training run. These metadata databases are usually private, and access is given selectively to members of a specific team or company. One exception is the OpenML database, which essentially shares the same capabilities described above but is open for anyone to contribute their ML-related metadata. [45].

Post-Creation Activities

In a research setting, the machine learning workflow of the researcher commonly concludes with the training and evaluation of the model. If the produced model achieves performance metrics superior to the state of the art, the results would be published for other researchers to

be informed and build upon. On the other hand, when applying machine learning techniques in a commercial setting, the goal is to incorporate the model's output in a product that customers (internal or external) will use. For this reason, the model artefact will have to be deployed so that it is reachable by the customer-facing part of the application. Various tools can facilitate serving machine learning models, such as TensorFlow-serving [46].

The final phase of the typical ML workflow is the continuous monitoring of deployed ML models to guard against the adverse effects of dataset shift, as explained in Section 2.2.

2.3.3 ML Platforms

To support data scientists throughout the typical ML workflow presented above, an organisation's software engineers develop and maintain ML platforms comprising multiple components [47, 48, 49, 40]. These components can include, for example, notebook servers used to conduct exploratory data analysis and training of ML models, ML model registries used to store trained ML models alongside training metadata, data warehouses for the storage of an organisation's raw data, feature stores that transform the raw data into usable features and ML model servers that are used to deploy models and respond to prediction requests.

Instead of developing and maintaining an ML Platform in-house, organisations could also choose to utilise a managed ML Platform solution from a third party. Relevant solutions from the three largest cloud providers are Amazon's *Sagemaker* [50], Microsoft's *Azure ML* [51] and Google's *Vertex AI* [52]. These cloud providers are uniquely positioned to offer managed ML Platforms, including ML monitoring features, to a large customer base that already uses their infrastructure to store data and execute ML workloads. Their products are, therefore, useful as a baseline for solutions that similarly target the ML monitoring domain. Regarding ML monitoring, the following are some of the technical limitations of these three managed solutions.

All three products offer covariate shift detection functionality, albeit with varying levels of support for customisation. *Vertex AI* exposes an API for creating/updating monitoring jobs that execute periodically. The algorithm used to detect covariate shift is fixed, namely *L*-

infinity distance for categorical features and *Jensen-Shannon divergence* for continuous ones [53]. Every feature is considered in isolation, so multivariate analysis is not possible. The user can parametrise the algorithms by setting a threshold per feature for the algorithm's output, above which covariate shift will be detected. Upon detection, the user can receive an email notification or have a message published to a message queue to which other services can subscribe. *Vertex AI* does not expose any mechanism for merging ground truth labels with the values predicted by the deployed ML models. Therefore, concept shift detection is not supported.

Azure ML offers very similar functionality to *Vertex AI*. In terms of covariate shift detection, there is a fixed number of metrics calculated per feature depending on its statistical type, such as *Wasserstein distance* for continuous features and *Euclidean distance* for categorical ones [54]. The main difference from *Vertex AI* is that covariate shift is not detected per feature but holistically by combining every feature metric into one scalar value for which the user can set a detection threshold. However, the algorithm for calculating this scalar value is undocumented, which may cause uncertainty when setting the detection threshold. Similarly to *Vertex AI*, Microsoft's product does not offer ground truth ingestion. Therefore, users interested in concept shift detection must build their own solutions to augment *Azure ML*.

Among the three products examined, *Sagemaker* is the most feature-complete. It offers similar functionality for detecting covariate shift using a fixed algorithm. Additionally, it supports ground truth label ingestion that enables the detection of concept shift using a fixed algorithm. The main difference compared to the other services is that it allows users to change the detection algorithms by providing their own container image. However, this extensibility mechanism does not facilitate an explicit separation between the tasks of data scientists and software engineers. The container image provided must follow a particular technical specification, such as reading and writing specific files based on environment variables. Implementing such technical details falls outside the domain of data science.

The following table summarises the features of the three ML platforms presented above:

Table 2.1: ML Platform Comparison

Feature	Vertex AI	Azure ML	Sagemaker
email notifications	✓	✓	✓
covariate shift	✓	✓	✓
custom algorithms	-	-	✓
ground truth ingestion	-	-	✓
concept shift	-	-	✓

An important weakness of the above platforms is that they fail to facilitate the collaboration between software engineers and data scientists that is required for the effective development of ML monitoring systems. Nahar et al. [6] report on the unique challenges that this collaboration brings. One reported challenge is the potential mismatch between the responsibilities assigned to a person and their capabilities and preferences. Specifically, data scientists prefer to receive support with software engineering tasks rather than doing it all themselves. On the other hand, software engineers find it challenging to perform ML tasks due to a lack of domain knowledge. Additionally, using different terminology in the two disciplines can lead to ambiguity, misunderstandings and inconsistent assumptions. Finally, they report that many of these conflicts stem from the lack of clear responsibility boundaries and recommend that teams carefully define them.

In this context, the following are some of the weaknesses that might limit the applicability of the reviewed ML platforms in the context of ML monitoring:

Vendor Lock-In In all three cases, the monitoring workflow definitions are vendor-specific. This means that in case an organisation wants to switch vendors, all monitoring workflows would have to be modified to fit the new vendor's API. From a vendor's point of view, there is little incentive to adopt open standards that allow compatibility with other vendor products, as that might lead to them losing customers to competitors. On the other hand, strongly coupling ML monitoring workflows to a single vendor's product poses commercial risks for client organisations.

Limited Customisability As discussed in Subsection 2.2.3, an integral part of a monitoring workflow is the selection of the dataset shift detection algorithm and the subsequent action taken. Two of the three products examined do not offer a built-in way for users to provide their own dataset shift detection algorithms or actions. Only Sagemaker provides a documented way for achieving this. However, it requires users to develop a containerised application that adheres to specific technical requirements, so it primarily caters to software engineers. For these reasons, all three services might have limited appeal to data scientists.

Unclear Boundaries As explained above, defining clear responsibility boundaries between software engineers and data scientists is paramount. None of the examined products achieves this as their usage requires a mixed software engineering and data science skillset.

2.3.4 ML-Related Challenges

While interest in ML is very high, translating a research achievement into a commercial success can be challenging. Organisations report that there are multiple obstacles deploying production AI/ML systems, with Delloite reporting "lack of technical skills" as a major challenge [38] and Gartner reporting that while launching AI/ML pilot projects is relatively easy, turning them into production systems is "notoriously challenging" [55].

These challenges are not a new phenomenon. Researchers from Google, an organisation that was an early adopter of ML in production, argue that ML systems are challenging to implement as they have all the challenges of traditional software systems in addition to ML-specific issues [42]. Additionally, they claim that in ML systems, only a fraction of the code is ML-specific, with the rest of the code concerning typical software system aspects such as data management or infrastructure configuration. This duality of ML systems often means their development needs multi-disciplinary teams comprising software engineers and data scientists. Nahar et al. [6] find that this collaboration between people from different disciplines brings unique challenges. One reported challenge is the potential mismatch between the responsibilities assigned to a person and their capabilities and preferences. Specifically, data scientists prefer to receive

support with software engineering tasks rather than doing it all themselves. On the other hand, software engineers find it challenging to perform ML tasks due to a lack of domain knowledge. Additionally, the use of different terminology in the two disciplines can lead to ambiguity, misunderstandings and inconsistent assumptions. The paper concludes that many of these conflicts stem from the lack of clear responsibility boundaries and recommend that teams should carefully define them.

2.4 Model-Driven Engineering

So far, this chapter has been concerned with the descriptive models of the ML domain. This section covers the different aspects related to prescriptive modelling in the context of software engineering.

From machine code to assembly, to systems programming in C, to object-oriented programming in Java, the history of computer programming can be viewed through the lens of abstraction. Over the years, software engineers have constructed a stack of abstraction layers to move further away from the inner workings of the hardware and closer towards the concepts they want to express. As computing became mainstream, increasingly more complex software systems started to be constructed. In the backdrop of ever-increasing software costs [56], the idea of utilising models to formalise and automate parts of the software engineering process started to gain ground. From the CASE tools of the 80s to contemporary model-driven engineering and low-code platforms, progress was not always straightforward and well-received. Similarly to the timeline of AI progress, there was, at times, a lot of excitement, inflated expectations, and disillusionment. The following are some of the milestones in the history of modelling applied in the software engineering context.

- Early Attempts: As evident by the term computer-aided software engineering (CASE), which alludes to the term computer-aided design (CAD), the intention was to adopt processes for designing and documenting software systems akin to other engineering disciplines such as mechanical engineering. The acronym CASE, for computer-aided software

engineering, first appeared in the early 1980s [57] to describe software packages meant to support software engineers in constructing software systems. As evident by the term, the vision was to recreate the success that computer-aided design (CAD) tools have enjoyed in other engineering disciplines such as aerospace and automotive. Some CASE tools mainly focused on graphical aspects such as flow or entity-relationship diagrams, but the most capable also included code-generation capabilities. The launch of the commercially successful EXCELERATOR greatly expanded the market for CASE tools, with sales growing at a rate of 70% at some points during the mid-80s. At the same time, partly due to inflated expectations created by the marketing departments of CASE tool vendors, these software packages drew a fair amount of criticism regarding low adoption [58] and unsubstantiated productivity gains [59, 60]. Nevertheless, CASE tools were an essential first step towards modelling being utilised in the context of software engineering.

- Standardisation efforts: A big problem with CASE tools was the lack of standardisation, which hindered interoperability between tools and the portability of the modelling artefacts. In 1989, the object management group (OMG) was founded to develop standards related to modelling in the software engineering domain. Indeed, in 1997, they publically released the unified modelling language (UML), which became a widely adopted standard for modelling software systems. The organisation has since published various related standards, such as MOF, BPMN, SysML and others [61].

- Proliferation of modelling approaches: With the success of UML and a more holistic approach compared to the one offered by the CASE tools of the 80s, modelling started to pick up steam again. In 2001, OMG released its vision of how its modelling standards should be jointly applied to software projects, termed model-driven architecture (MDA). MDA focuses on separating business logic from platform-specific details by using platform-independent models that are to be transformed into platform-dependent models and can be used for generating code artefacts. This approach seeks to reduce manual programming effort and improve portability between platforms [62].

  Similar ideas have also been the subject of academic research. In the software world, the

terms model-driven software engineering (MDSE) and model-driven development (MDD) advocate for similar workflows that elevate the importance of models, model transformations and code generation without necessarily relying on OMG standards. A more general term considered a superset of MDSE and MDD is model-driven engineering (MDE). MDE's scope goes beyond the modelling of only pure software artefacts to cover more aspects of the engineering process, such as the modelling of cyber-physical systems [62]. For the remainder of this thesis, the term MDE will be the term used to refer to the application of modelling in the software domain.

- Industry Adoption: Over the years, MDE gained traction in various industries, including automotive, aerospace, IT, and defence [63]. Many companies have adopted MDE principles and tools to improve software development productivity, quality, and maintainability. Open-source MDE frameworks and modelling tools, such as Epsilon and Eclipse Modeling Framework, became widely used.

- Ongoing Development: MDE continues to evolve with ongoing research and advancements in modelling techniques, languages, and tools.

2.4.1 Modelling Fundamentals

In the context of MDE, an essential concept is that of metamodelling. As the etymology of the term suggests, metamodelling is the modelling of models. In fact, a hierarchy of modelling layers can be built this way, with each layer modelling the one below. The standard approach, is to have three modelling layers (M1-M3) built on top of each other [62]. The bottom modelling layer (M1) models physical or digital objects in the real world (M0). The highest layer (M3) can self-referentially model itself, thus closing the loop. Figure 2.4 shows a simple example of using this modelling hierarchy to model a physical object. At the top, there is OMG's metamodelling language, MOF, which can be used to define metamodels, which are also called modelling languages. Below that is UML, a modelling language that can be used to create models. In the base modelling layer, there is a model which conforms to the UML specification and which, in turn, describes the physical object that is to be modelled.

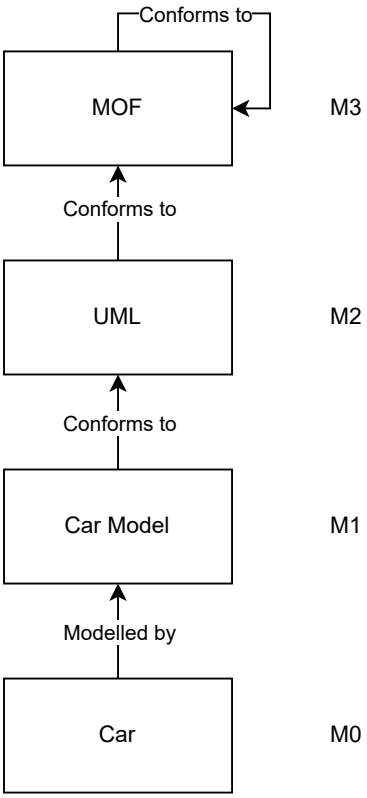


Figure 2.4: M0-M3 Modelling Layers

2.4.2 Language Definition and Development Techniques

As mentioned above, models in the M2 layer can be seen as modelling languages that can be used to create models in the M1 layer. This sections covers the different kinds of modelling languages and certain aspects of their development.

General and Domain-Specific Languages

Modelling languages can be broadly categorised into two types: General Modelling Languages (GMLs) and Domain-Specific Modelling Languages (DSMLs or DSLs) [62].

GMLs are designed to be applicable across a wide range of domains and problems. They provide a broad set of constructs and semantics that can be employed in multiple contexts. The versatility of GMLs can be helpful, but their broad scope can also lead to unnecessary complexity for specific tasks.

DSLs, on the other hand, are tailored for specific application domains or problem areas. They

encapsulate domain knowledge and provide constructs that are directly aligned with the concepts and abstractions of that domain. The main trade-off is the ease of use for a specific domain but, at the same time, unsuitability for general-purpose modelling.

Modelling Language Syntax

As described above, a modelling language can be defined using a metamodelling language, but this only covers the language's abstract syntax. The abstract syntax defines the conceptual structure of models. It represents the core entities, their attributes, and the relationships between them without defining how these entities are concretely represented [62].

On the other hand, the concrete syntax of a modelling language deals with the tangible representation of models. It specifies how the concepts defined in the abstract syntax will be presented. Based on the type of concrete syntax, a modelling language can be textual, graphical, or even have both types of concrete syntax [62].

Standalone and embedded DSLs

An interesting distinction that can be made mainly for textual DSLs is between standalone/external and embedded/internal DSLs [64].

A standalone DSL has its own syntax and tools for parsing, compiling, and execution. It is designed from the ground up for a specific domain and does not rely on the constructs of another general-purpose language.

An embedded DSL, on the other hand, is hosted within a general-purpose programming language. It leverages the host language's syntax, semantics, and infrastructure, providing domain-specific constructs as libraries or extensions to the host language.

2.4.3 Model Management

In addition to defining models in the various layers of the modelling hierarchy, an integral part of the MDE process is performing various tasks on these models to gradually turn them into a concrete solution for the target domain. These tasks are often defined for models at the M2 layer and applied in every model in the M1 layer that adheres to the M2 layer model. This section covers some of the most common model management tasks.

Model Validation

While metamodeling languages provide mechanisms to define certain structural constraints for downstream models, it is possible that the metamodelling language used might not be enough to express all requirements for a valid model.

Addressing this problem, model validation languages, such as OCL or EVL, are designed to express more complex constraints. Using these model validation languages, one can construct invariants for types in the M2 layer that must hold true for any instance of the type found in an M1 layer model for it to be valid.

For example, for an M2 model that contains the type *Person* with a String attribute called *name*, we could define an invariant that requires the *name* attribute to always be non-empty. Then, for every *Person* instance in an M1 model that adheres to this metamodel, the *name* attribute must be non-empty for the model to be valid.

Model-To-Model Transformation

A Model-to-Model (M2M) transformation is a process that produces one or more target models from one or more source models. As shown in Figure [TODO], transformations are specified at the metamodel layer and applied at the model layer [cite]. A popular method of specifying M2M transformations is by using rule-based languages, such as ATL [cite] and ETL [cite]. The target and source models can adhere to the same or different metamodels. In cases where the

target and source models adhere to different metamodels, the transformation is classified as exogenous. On the other hand, if the source and target models adhere to the same metamodel, the transformation is classified as endogenous.

An example scenario where M2M transformations play an important role is in MDA. As described in the introduction of this section, the MDA approach includes a step where platform-independent models are mapped onto platform-specific models in order to gradually get closer to a concrete solution for the problem domain. This mapping can be achieved by specifying an M2M transformation.

Model-To-Text Transformation

A Model-to-Text (M2T) transformations is a process that produces text from a model. An important use case for M2T transformation is the generation of code but other text can also be generated such as documentation, test cases or deployment scripts. While M2T transformations can be implemented in a general-purpose programming language, domain-specific languages, such as EGL [cite], can also be used. The advantage of using a DSL for implementing M2T transformations is ease of use. In the case of EGL, for example, its template-based nature make it easier to mix the static and dynamic parts of the generated text while keeping the text generator scripts highly readable.

An example use case of M2T transformations can once again be found in the MDA paradigm. Once the platform-specific model has been produced, an M2T transformation can be leveraged to generate code and accompanying artefacts, such as deployment scripts, that constitute a concrete solution that can be used by end-users.

2.4.4 Modelling in Practice

For the application of model-driven engineering in practice, the community has developed a vibrant ecosystem of open-source tools. Two such tools that have been extensively utilised in the context of this thesis are EMF and Xtext.

EMF

The Eclipse Modeling Framework (EMF) [65] is essential for implementing an MDE approach. It offers tools for structured data modelling and code generation. It bridges design and implementation, facilitating model-driven application development.

The key features of EMF that have been utilised in this thesis are the following:

Metamodelling Central to EMF is ECore, an M3 metamodelling language which started as a separate effort from MOF but has been developed into MOF's de facto implementation.

Serialisation and Deserialisation EMF provides support for serialising and deserialising model instances to and from various formats, including XML-based formats like XMI (XML Metadata Interchange). This enables EMF users to transfer models across different tools or services in a network.

Code Generation EMF can automatically generate Java code from Ecore models. This generated code includes classes for representing domain model elements in-memory. These in-memory models can be are compatible with the serialisation and deserialisation mechanism described above.

Xtext

Xtext is a framework that facilitates the creation, editing, and processing of textual DSLs, offering a comprehensive environment for language definition, editing, code generation, and tool integration [66].

The central component of Xtext is its grammar language, which closely follows the Extended Backus–Naur Form (EBNF) and can be used by users to define the concrete syntax of textual DSLs. Given a user-created grammar file, the framework can automatically generate a parser

that processes text files and outputs EMF models. These models conform to EMF metamodels that can be explicitly provided by the user or automatically derived from the grammar rules.

In addition to parser generation, Xtext provides a number of convenience features that streamline the process of creating textual DSLs. For example, the framework supports the automatic generation of language editors with syntax highlighting, error highlighting and autocomplete support. The language editor can be generated as an Eclipse plugin, an LSP-compatible language server, or a web-based service. Furthermore, the generated code that supports the editor's feature is easily extensible by the user to further customise the editing experience with modifications such as custom model validation logic.

2.5 Chapter Summary

This chapter presented a review of background material in the fields of machine learning and model-driven engineering. With regards to machine learning, the chapter covered the topic of dataset shift which is a phenomenon that can degrade the performance of ML models. In addition, various techniques found in the literature that can mitigate the negative effects of dataset shift were covered and a common pattern between them was identified. The chapter also covered the application of machine learning techniques in industry and the commercial platforms that are available for this purpose along with their weaknesses that make them unsuitable for the task of monitoring against dataset shift. With regards to model-driven engineering, the chapter covered the fundamental concepts along with two commonly-used tools that were utilised for this thesis.

Chapter 3

Architectural Overview

This chapter presents an overview of the architecture for the proposed solution to address the challenges identified in Chapter 2. The solution is named *Panoptes*, a Greek adjective that means all-seeing, inspired by the fact that it monitors ML models and checks whether they perform well over time.

The chapter also introduces an illustrative example, which is used as an aid in presenting the proposed solution in the rest of the thesis. The example describes a scenario in which the performance of a deployed ML model might be negatively affected by dataset shift, and therefore, implementing a monitoring workflow would be beneficial.

The rest of the chapter is organised as follows. Section 3.1 gives the context in which the proposed solution was developed. In Section 3.2, the aforementioned illustrative example is introduced. Section 3.3 presents the scope of the work presented in this thesis. Section 3.4 provides a high-level overview of the proposed solution. Finally, Section 3.5 cover the solution's major design decisions.

3.1 Context

The presented work was undertaken within British Telecom's (BT) applied research department. BT is the largest telecommunications provider in the UK, providing telephony, fixed broadband and mobile services to fourteen million households and employing more than ninety-nine thousand people [67]. Similarly to other large enterprises, BT is interested in leveraging ML models to attain a competitive advantage in the market. However, due to complex interactions between algorithms and data, ML models can fail unexpectedly, potentially exposing their operators to adverse economic and legal consequences. Wanting to hedge against these risks, BT has shown strong interest in the area of *AI governance*, which can be defined as a system of rules, practices, processes, and technological tools that ensure the responsible usage of AI technologies [68]. Attesting to BT's interest is its involvement in the TM forum alliance[69], which resulted in the publication of standardised guidelines regarding AI governance. The proposed approach addresses one aspect of AI governance: continuously monitoring deployed ML models to ensure they perform as expected.

3.2 Illustrative Example

In order to provide a clearer understanding of the domain of ML model monitoring, let us delve into a simple hypothetical scenario. This scenario revolves around a customer support call centre, where customers can seek assistance for issues they encounter with a purchased product. Each incoming call follows a series of stages: Initially, the customer is placed in a waiting queue until a customer support representative becomes available to assist them. Subsequently, they converse with the representative, who attempts to resolve their issue. Finally, the call concludes whether or not the customer's problem was resolved.

For this example, let us imagine that the company's marketing department is interested in determining the satisfaction levels of customers who contact the customer support call centre. Armed with this information, they aim to offer appropriate compensation to dissatisfied customers to enhance the company's reputation. However, since it is impractical to contact

every customer and directly inquire about their customer support experience, the marketing department would like a dashboard implemented that shows the predicted satisfaction of each customer based on the recorded information from each call.

From a data science perspective, this scenario aligns with a typical supervised learning classification task. As discussed in Section 2.1, such tasks require the collection of labelled samples, which are then used to train an ML model. In this case, the labelled samples consist of the recorded information from each call and whether or not the customer was satisfied with the service. For our example, let us assume that the recorded information includes the duration of the customer's wait in the queue, the duration of the call with the customer support representative, and whether or not the issue was resolved successfully. These three values can be viewed as a three-dimensional vector variable denoted as X. The corresponding label can be represented as a random variable Y that takes on values from the set $\{0, 1\}$ for unsatisfied and satisfied customers accordingly. Consequently, the trained ML model can be seen as a mapping from X to Y derived from the training dataset. This mapping enables the prediction of future customers' satisfaction based on the information recorded during their calls.

Upon completing the initial training of the ML model, the company's data scientists encounter a challenge. Despite achieving good accuracy on the test set during the training phase, there is no guarantee that the same level of accuracy will persist in the future. As discussed in Section 2.2, this uncertainty arises due to dataset shift, wherein the joint probability distribution of variables X and Y might change, potentially leading to a negative impact on the accuracy of the trained ML model.

In the context of the call centre example, the following are some of the scenarios that can lead to the manifestation of the types of dataset shift:

3.2.1 Covariate Shift

On a particular day, an unusually high number of workers might be on leave, resulting in increased customer waiting times. Assuming that customers' preferences regarding waiting

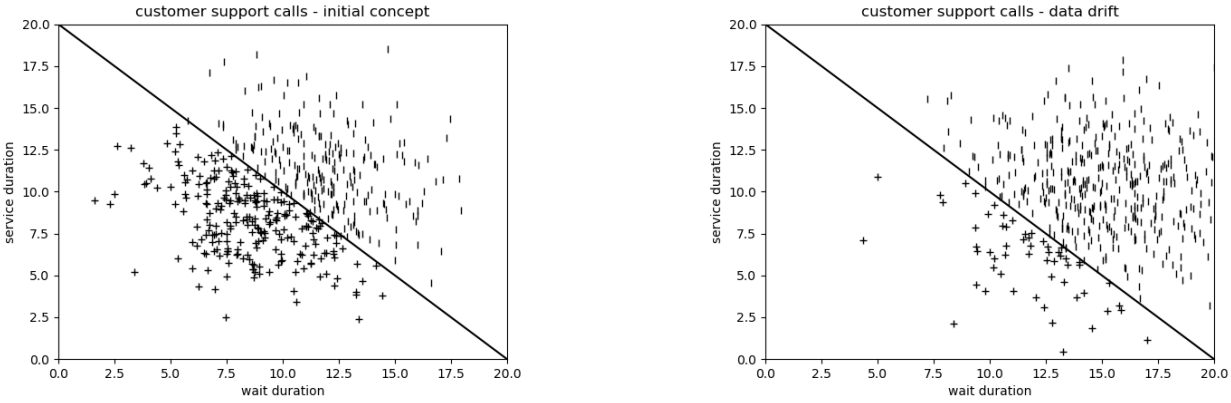
times remain constant, more customers will be left unsatisfied. According to the terminology of dataset shift, this scenario exemplifies covariate shift.

Figure 3.1 visually depicts covariate shift for our example scenario. For ease of exposition, we assume that customer satisfaction is deterministic, with customers being satisfied when the sum of the wait and service duration is under twenty minutes and unsatisfied otherwise. Additionally, we assume that issue resolution does not affect customer satisfaction, so that it can be ignored in the figure. In both subfigures, satisfied customers are represented by a + sign, while unsatisfied customers with a | sign. Additionally, a solid line shows the optimal decision boundary that perfectly classifies customer satisfaction. In the left subfigure, wait and service duration follow a normal distribution with a mean value of ten minutes and a standard deviation of one minute. The right subfigure shows that the distribution of wait duration values has shifted to now follow a normal distribution with a mean value of fifteen minutes and a standard deviation of one minute. This shift has resulted in more data points falling above the unchanged decision boundary.

In theory, covariate shift should not affect the performance of the ML model, as the input-to-output mapping remains constant. However, when the ML model is relatively simple (e.g., a linear classifier), it may accurately approximate the true mapping within a specific range of input values but produce incorrect results when the input values deviate from that range [25]. In situations where it is known that a model's performance deteriorates when the input falls outside a particular range, identifying covariate shift can be advantageous, especially as it does not require the collection of additional ground truth labels.

3.2.2 Concept Shift

Over time, customers might become more tolerant towards extended waiting times. Consequently, even if waiting times remain constant, an increased number of customers will be satisfied. In other words, the mapping between the input and output has changed. In the terminology of dataset shift, this phenomenon represents concept shift. Detecting this type of



(a) Initial Scenario

(b) Covariate Shift

Figure 3.1: Covariate Shift

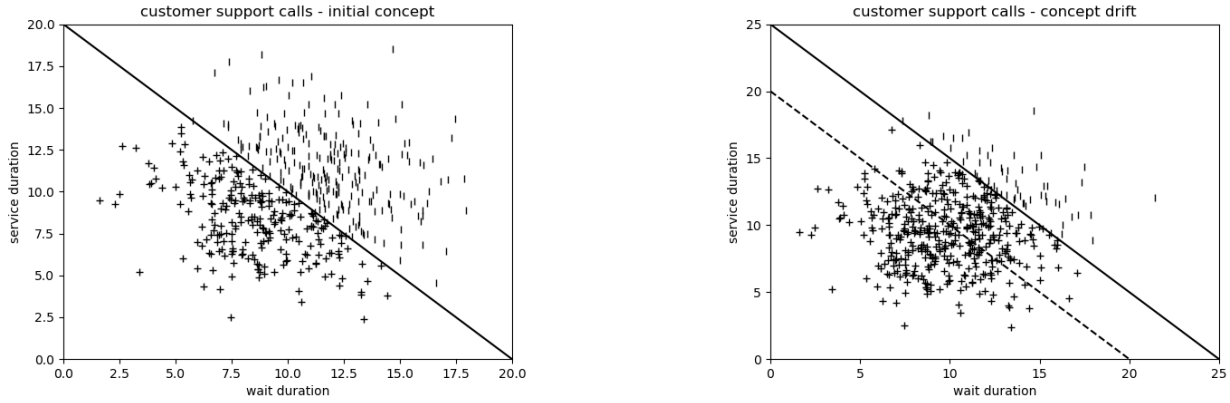
dataset shift necessitates the acquisition of ground truth labels, as no observable change occurs in the distribution of input values.

Figure 3.2 visualises concept shift for our example scenario. The left subfigure once again shows the initial distribution of wait and service times and customer satisfaction. The right subfigure shows the same variables after customers' attitudes towards waiting and service times have changed. Here, in addition to the optimal decision boundary shown with a solid line, the previous decision boundary is also shown with a dashed line. We can see that customers are now satisfied when the sum of wait and service duration is below twenty-five minutes. As a result, even though the distribution of wait and service duration values have remained constant, there are more satisfied customers.

Compared to the covariate shift scenario described above, the presence of concept shift will be more challenging to detect as there are no changes in the input data. In order to detect the fact that customers' preferences have changed, we will have to gather their feedback after they have been served, which could add complexity to our monitoring system.

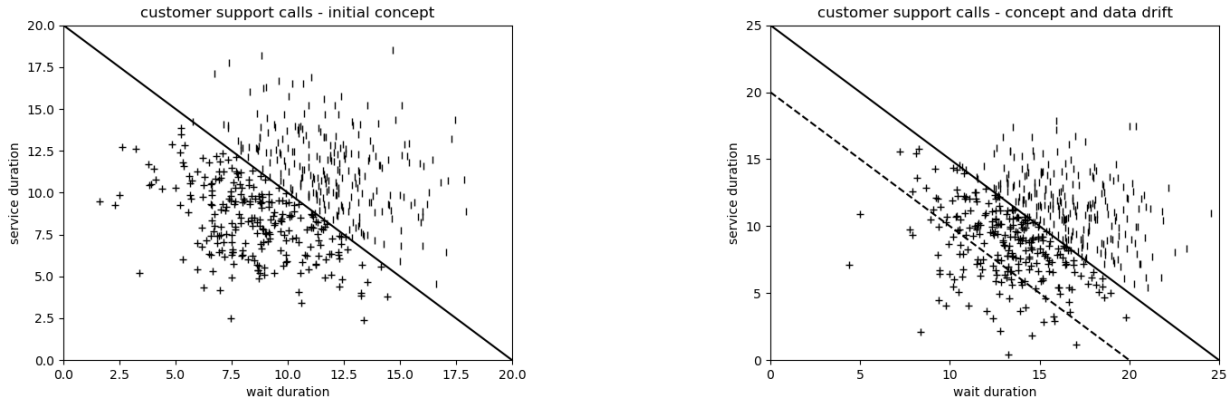
3.2.3 Other Types of Shift

The presence of covariate shift does not preclude concept shift and vice versa; the two types of dataset shift can occur at the same time. In other words, the average waiting time could



(a) Initial Scenario (b) Concept Shift

Figure 3.2: Concept Shift



(a) Initial Scenario (b) Covariate & Concept Shift

Figure 3.3: Other Types of Shift

increase while customers become less tolerant of lengthy waiting periods. This scenario falls under the "other types of shift" category in the dataset shift terminology.

Figure 3.3 is a visual depiction of this type of dataset shift for our example scenario. As we can see, compared to the initial scenario on the left subfigure, both the distribution of wait duration values and the decision boundary have changed.

3.2.4 Monitoring Solution for the Call Centre Example

In order to prevent potential performance degradation caused by any of the dataset shift scenarios described above, the data scientists of the example would like to implement a monitoring

workflow. As discussed in Section 2.1, implementing a dataset shift monitoring workflow can be split into several phases. For simplicity, the following monitoring workflow, which monitors against covariate shift, is examined: Whenever a set of one hundred calls is logged (standby phase), an algorithm shall be executed which compares the distribution of the call duration variable from the log to that found in the ML model's training set (algorithm execution phase). If the algorithm execution detects covariate shift, an email notification is sent to the data scientist responsible for this ML model so they can investigate further (action phase).

3.3 Scope

For a solution to address the challenges posed by the example introduced in the previous section, multiple elements would have to be considered. In the scope of this thesis, some of these elements might be less relevant. To this end, this section clarifies the scope of the thesis and points out select out-of-scope elements.

Target Audience. The primary target audience for the developed solution is professionals with expertise in statistical modelling but without software engineering skills necessary for large-scale ML deployments, such as networking and cloud computing. This group of professionals will be referred to as *data scientists* throughout this thesis. Our primary goal is to enable them to deploy ML monitoring workflows autonomously. For this reason, we steer the design and evaluation phases based on their feedback. At the same time, integrating the developed solution with an underlying ML platform requires some software engineering effort. Care will be taken to ensure this integration does not require specific MDE knowledge to keep it in line with typical software engineering tasks, but measuring integration effort is considered outside the scope of this thesis. Additionally, if a one-time integration effort automates multiple future ML monitoring workflow deployments, it might be a worthwhile endeavour.

Dataset Shift Detection Algorithms. Selecting the appropriate algorithm or even developing a novel one for detecting dataset shift is a fundamental component of an ML monitoring

workflow. This is considered the domain of data scientists, and this thesis does not aim to make any contributions. On the other hand, part of our extensibility goal is to support as many such algorithms as possible.

Machine Learning Scenario Support. As discussed in Section 2.1, theoretical analysis of machine learning mainly focuses on the supervised learning scenario. Therefore, no claims about the solution's suitability for other scenarios are made.

Infrastructure Portability. While the developed execution environment aims to be portable between computing infrastructures and interface with as many ML platforms as possible, it does have some requirements. As we will present in Chapter 3, these are quite minimal, and we consider it reasonable to expect that most platforms can meet them. At the same time, evaluating this by porting to multiple environments was considered outside the scope of this thesis.

Authentication and authorisation. In a professional setting with teams of multiple people responsible for multiple deployed ML models, it would be essential to implement authentication and authorisation mechanisms to enable effective and secure deployment of ML monitoring workflows. On the other hand, this feature was considered outside the scope of this thesis as it would contribute little to its academic merits.

3.4 High-Level Overview

This section provides a high-level overview of the proposed solution. Technical details provided for each component will be coarse as finer details are provided in Chapters 3 and 4. As already discussed in Chapter 2, implementing monitoring workflows is a complex technical project that requires the interaction of different stakeholders. To elucidate how the proposed solution can streamline the process of monitoring workflow implementation, Figure 3.4 shows how the

various components interface with each other and the responsibilities of relevant stakeholders. There are three architectural components: The ML platform, the Panoptes orchestrator and the platform-specific components. This thesis focuses on three main stakeholder groups that are relevant to the proposed solution, instead of the more comprehensive list presented in Section 2.3.1 that also includes non-technical stakeholders. The following paragraphs contain a brief description of the responsibilities of each stakeholder group.

Language Engineers: These are software engineers who specialise in developing DSLs, primarily using model-driven engineering techniques. In the context of Panoptes, the author has assumed this role and developed the Panoptes Description Language (PDL) and the accompanying execution environment in the form of the Panoptes orchestrator.

Data Scientists: Occupying a central role, data scientists leverage the ML platform for training and deploying ML models. Using the provided DSL, they can also specify high-level models of monitoring workflows, which the orchestrator subsequently ingests.

Software Engineers: Software engineering are responsible for implementing and maintaining both the ML platform and the platform-specific components. Their role ensures that the underlying infrastructure is robust, scalable, and resilient.

More in-depth details about the components shown in Figure 3.4 are provided below.

3.4.1 ML Platform Layer

ML platforms are vital for organisations wishing to use ML to improve their operational efficiency or enhance their products. As such, they exist independently of the other components shown in Figure 3.4. Despite that, it is included in the discussion of the proposed solution as it is so critical. As explained in Section 2.3, an organisation's software engineers develop and maintain ML platforms that contain all of the components data scientists need to train and

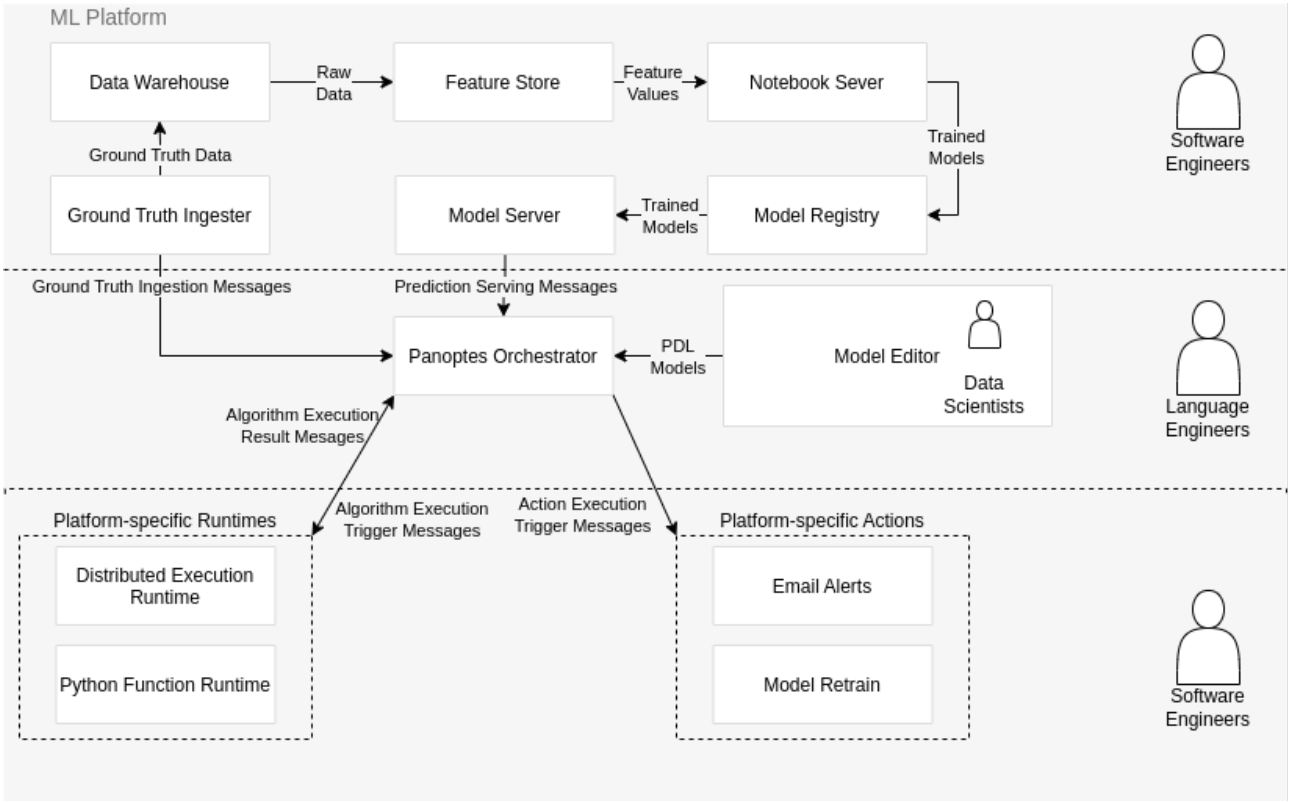


Figure 3.4: High-Level Architectural Overview

deploy ML models. The hosted ML models can then serve prediction requests from various applications.

In the context of the example scenario, the ML platform is used to train the ML model, which predicts the customers' satisfaction. This ML model is subsequently used to serve the prediction requests made by the marketing dashboard application used by the organisation's marketing department.

3.4.2 MDE Layer

In the middle section of Figure 3.4 lay the components for which language engineers are responsible. The central component of this layer is the orchestrator that, as the name suggests, is responsible for orchestrating the phases of the monitoring workflows that data scientists would like to implement. For the high-level specification of a monitoring workflow, data scientists can use the Panoptes Description Language (PDL), a DSL designed explicitly for this purpose. The information in a PDL model informs the orchestrator of the sequence of actions to implement

the monitoring workflow correctly.

In the example scenario, the monitoring workflow progresses from the standby phase to the algorithm execution phase whenever one hundred new calls are logged. For the orchestrator to know when that happens, it needs to interface with the ML platform. To cover all scenarios, a monitoring workflow can enter the algorithm execution phase under different conditions as well. Chapter 4 covers these conditions in detail.

The orchestrator is not capable of executing dataset shift detection algorithms itself. Instead, it interfaces with the algorithm runtime services, which are part of the platform-specific components and are responsible for algorithm execution. After the algorithm specified in the PDL model is executed, the result is fed back to the orchestrator. This brings the monitoring workflow to the action phase. To successfully conclude this phase, the PDL model specifies which action, if any, needs to be executed depending on the algorithm execution result. Once again, the orchestrator cannot execute actions itself but interfaces with the action services, which are part of the platform-specific components and are responsible for action execution.

3.4.3 Integration Layer

This layer contains platform-specific components that are responsible for the actual execution of a monitoring workflow's phases. Two different kinds of services are part of the platform-specific components: algorithm runtime and action services. As the names suggest, these are responsible for the algorithm execution and action phases of a monitoring workflow, respectively.

Algorithm runtime services follow the function-as-a-service (FaaS) philosophy. This application development and deployment model has gained popularity recently and is offered by major cloud providers (e.g. AWS Lambda, Google Cloud Functions, and Azure Functions). With FaaS, developers implement their applications' business logic in one of the languages and web frameworks supported by the providers' FaaS runtime, typically in the form of a function with a specific input/output signature as the runtime requires. The runtime then handles lower-level technical concerns like server provisioning, request routing, and load balancing.

In the case of Panoptes, the target audience of algorithm runtimes is data scientists. Similarly to the FaaS model, data scientists would like to implement dataset shift detection algorithms at the logical/mathematical level without being involved with lower-level technical details such as the retrieval from the ML platform of the data necessary to execute the algorithm. By utilising a Faas-style algorithm implementation and the information provided by the orchestrator, algorithm runtimes have all the necessary information to retrieve the correct data, execute the algorithm and feed the result back to the orchestrator.

After receiving the result of the algorithm execution, the orchestrator checks whether an action needs to be executed based on the PDL model of the monitoring workflow. Similarly to the execution of algorithms, action executions are handled by action services. The main difference is that for action executions, data scientists do not need to provide any part of the implementation. Despite that, action executions can still be parametrised as needed. In the case of the call centre scenario, for example, an action service that handles email notifications would need the recipient's email address.

More details about the information that the platform-specific components receive are provided in Chapter 4. Additionally, Chapter 5 provides more technical details in the context of the reference implementation of Panoptes.

3.5 Design Decisions

This section covers the major design decisions and how they contribute to the desired quality attributes.

3.5.1 Component Interfacing

In order to achieve portability and extensibility, Panoptes has been designed on the basis of the service-oriented and event-driven architectural patterns. The following is an explanation of how these architectural patterns aid in achieving the stated goals:

In a service-oriented architecture, a system is implemented as a collection of software components, known as services, that communicate with each other over a network. Each service provides a specific functionality and can be accessed and used independently by other services or external applications. Additionally, each service exposes its functionality through a standardised interface and hides its implementation details. This aids in the loose coupling of the services and makes it feasible to replace existing services or add new ones without affecting the overall system.

The loose coupling attribute that the service-oriented architectural pattern offers, make it suitable for Panoptes' needs. An organisations' software engineers can add new functionality in the ML Platform and integration layers in the form of new services, without having to be concerned about the inner workings of the MDE layers, thus enabling extensibility. Additionally, the services that constitute the MDE layer can be deployed in different environments, such as cloud or on-premise infrastructure, which makes the solution portable.

In the context of Panoptes, the event-driven architectural pattern is also critical and complements the service-oriented architectural pattern well. At its core, an event-driven system is predicated on producing, detecting, consuming, and reacting to events. In this context, an event refers to a change in state or a significant occurrence within a system. For example, logging a new call would be an event in the call centre scenario. This paradigm facilitates systems to respond in real-time to specific triggers, promoting decoupling between components, enhancing flexibility and ensuring the system is robust to evolving needs.

From a technical point of view, the following are the main points of an event-driven architecture:

Message-based: Events are serialised in a specific format to be passed between various services.

Producers: Services that generate or produce events. In the call centre example, a call logging service would be an event producer.

Consumers: Services that are interested in specific events and react to them. They listen for and handle the events they are interested in. In the call centre example, the orchestrator would be an event consumer.

Event channel: A system that transports the events from producer to consumer. This can be implemented in various ways, including message queues, enterprise messaging systems, or even simpler constructs like in-memory queues. This design allows the producer of an event to be agnostic towards the consumer; it just emits the event. Likewise, the consumer does not need to know who the producer is; it just listens for the event.

The adoption of event-driven architecture further enhances the portability and extensibility of Panoptes. The orchestrator can consume and produce a specific set of events in a standardised format. As long as the message format is followed, it becomes feasible to substitute individual components without changing the rest of the system. This also means the orchestrator can be deployed on different computing infrastructures or cloud providers with minimal friction. Extensibility is also enhanced as additional algorithm runtime and action services can be added as long as they can consume the events produced by the orchestrator and produce the events that the orchestrator expects.

3.5.2 Panoptes Description Language

Developing a DSL is a strategic decision in our proposed solution to facilitate the distinct roles of data scientists and software engineers. A DSL caters specifically to a particular domain or area of interest, and in our context, it targets the domain of ML model monitoring.

The key benefit of implementing a DSL in this scenario is its ability to support the specification of monitoring workflows at a high level of abstraction. Instead of burdening data scientists with the intricate details of the software implementation or the underlying infrastructure, a DSL abstracts these complexities. Data scientists can define the requirements of their monitoring workflows using familiar concepts. This reduces the learning curve and allows data scientists

to express their monitoring logic concisely and accurately.

In terms of concern separation, offering a DSL allows Panoptes to erect clear boundaries between the responsibilities of data scientists and software engineers. Data scientists are empowered to focus solely on the domain logic – defining how dataset shift should be detected, the detection thresholds, and the desired corrective actions. Conversely, software engineers can concentrate on the ML platform's technical aspects, ensuring, for example, that it achieves the required performance levels. This separation simplifies the workflow and reduces the potential for miscommunication.

A stable DSL can also offer inherent portability. Once a monitoring workflow model is constructed using the DSL, its logic remains consistent even if there is a change to the other components of the system. This means that data scientists do not need to reimplement their monitoring workflows when software engineers need to migrate the solution across different ML platforms or infrastructures.

3.5.3 Separation of Orchestrator and Platform-specific components

The orchestrator stands at the centre of the proposed solution. It serves as an intermediary, bridging the gap between high-level monitoring workflow specifications by data scientists, as described in the DSL, and the platform-specific components that ensure the execution of these specified tasks. The decision to architect Panoptes as a set of distributed services instead of a monolith is essential for achieving the goals of portability, separation of concerns, and extensibility.

Separation of Concerns

The orchestrator is instrumental in separating concerns between data scientists and software engineers. The previous subsection mainly focused on the benefits data scientists receive from using the DSL. Complementary to that are the benefits software engineers receive from the

separation of concerns that the orchestrator, coupled with the DSL, enables. Since the orchestrator produces a standardised set of event messages documented by the language engineers in charge of the orchestrator, the software engineers only need to implement web services that respond to those without needing to delve into the details of the ML monitoring domain or have experience with MDE and DSLs. This arrangement means that data scientists are insulated from the underlying technical complexities of executing these workflows, allowing them to remain focused on the scientific and algorithmic aspects. Conversely, software engineers, who are more attuned to these complexities, can channel their expertise into developing and refining the platform-specific components without needing to delve into the intricacies of the ML model or its monitoring workflows.

Portability

The orchestrator's design is intentionally decoupled from any specific ML platform or platform-specific components. This modular design ensures that the core logic and functionality of the orchestrator remain consistent, regardless of the underlying infrastructure.

For instance, consider a scenario where an organisation decides to migrate from one cloud provider to another, or even from a cloud-based solution to an on-premises one. In such cases, the orchestrator remains unaffected. As long as the new ML platform and accompanying platform-specific components adhere to the standardised interface, all monitoring workflows can continue to be executed without any necessary modifications. This flexibility reduces engineering effort and ensures a smoother transition between computing infrastructures.

Extensibility

The modular nature of the orchestrator and the platform-specific components contribute to the solution's extensibility. On the one hand, software engineers can easily support new languages for implementing algorithms by developing additional algorithm runtimes. Likewise, as data scientists innovate and develop new dataset shift detection algorithms, they can seamlessly inte-

grate these into monitoring workflows, confident that the orchestrator and algorithm runtimes will manage their execution.

3.6 Chapter Summary

This chapter has presented an architectural overview of the proposed solution for monitoring ML models, focusing on effective collaboration between data scientists and software engineers, enabled by the work of language engineers. The architecture contains three dictinct layers, the ML platform layer, the MDE layer and the integration layer. Through the deliberate design decisions documented in this chapter, the solution achieves the three desired quality attributes of portability, extensibility and separation of concerns that are set as the objectives of this thesis. Building on this chapter, Chapter 4 covers how PDL expresses the different aspects of the modelled domain and Chapter 5 covers the technical considerations that went into the reference implementation of Panoptes.

Chapter 4

Panoptes Description Language

This chapter introduces the Panoptes Description Language (PDL). This is the DSL specifically designed for the domain of ML monitoring workflows. Both the PDL metamodel and its textual syntax are covered.

4.1 Core

Before discussing anything related to dataset shift, some core classes must be introduced to the DSL's metamodel. These classes lay the foundation by representing the available ML models, the inputs used to train them, their outputs, and whether they are currently deployed and serving prediction requests. Figure 4.1 shows the classes discussed in this subsection.

The first core class is *Platform*. A *Platform* instance is the top-level element in a PDL model. It represents the infrastructure that hosts all ML-related components. A *Platform* directly contains instances of *Model*, *FeatureStore*, *AlgorithmRuntime*, *Algorithm*, *Action* and *Deployment*. An ML platform can be utilised to deploy one or more ML models in production. Over time, as more data become available and more advanced ML techniques are developed, data scientists might choose to replace an older ML model with a newer one. This can be hidden from any downstream service that consumes the predictions of an ML model.

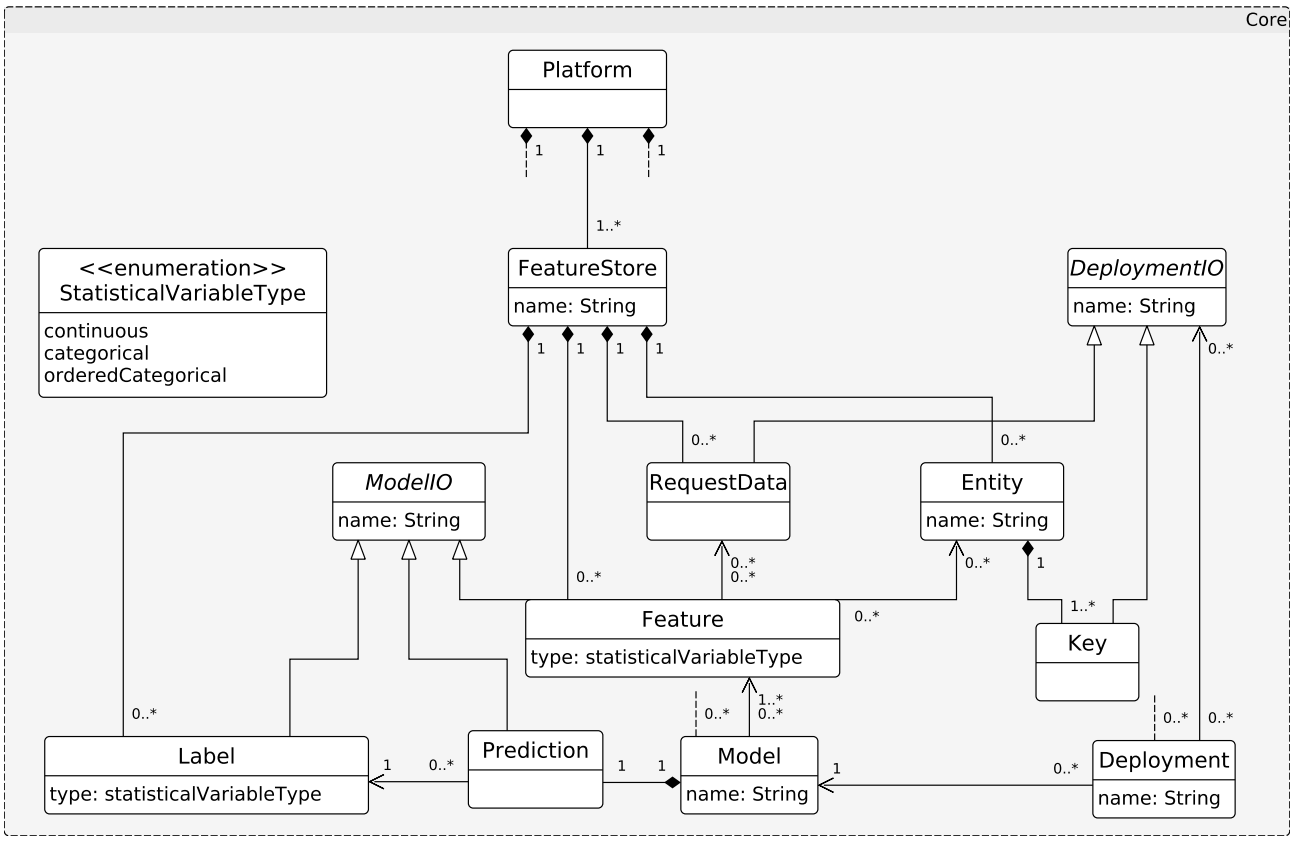


Figure 4.1: Core classes of the PDL Metamodel.

A distinction has been made between the *Deployment* and *Model* classes to accurately represent this. *Model* instances represent ML models regardless of whether they are currently being used in production or not. A *Model* references one or more *Features* and a *Prediction* to express an ML model's inputs and output, respectively. Additionally, a *Prediction* references a *Label* that represents the ground truth values the ML model tries to predict. Since the *Feature*, *Prediction* and *Label* classes are all related to the input and output of ML models, they subclass the abstract class *ModelIO*.

On the other hand, *Deployment* instances represent a specific task that can leverage an ML model, such as predicting customer satisfaction in the call centre example. Each *Deployment* references a *Model* instance, representing the current ML model used to accomplish the task.

Finally, the concept of a feature store is introduced to express the fact that multiple ML models can use each feature, and those ML models might attempt to predict the values of the same variable. Feature stores are represented in the metamodel by the *FeatureStore* class whose instances contain multiple *Feature* and *Label* instances so that they can be referenced by as

many *Model* instances as needed.

With the metamodel classes introduced so far, an initial model of the call centre scenario can be constructed. In the rest of the chapter, PDL models will be presented in a textual concrete syntax that was developed using Xtext [66]. The full grammar of this concrete syntax is available in Appendix A. As will be discussed in chapter 5, the overall system is architected in such a way that the concrete syntax of PDL is loosely coupled with the rest of the system so that adding an additional concrete syntax is relatively easy.

Listing 4.1 shows the initial PDL model for the call centre scenario Model elements are defined by the name of their class, followed by the name of the instance and brackets that include the attribute values and references of the instance. In this example, Lines 1-5 define a *Model* instance named *callcentreDecisionTree*. Line 2 specifies the *Features* that this *Model* references using the keyword "uses", line 3 defines the *Model's Prediction* using the keyword "outputs". Line 4 specifies the *Label* that the *Prediction* references using the keyword "predicts". Lines 7-14 define a *FeatureStore* instance, with lines 8-11 defining the three *Features* referenced by the afformentioned *Model* using the keyword "features". Lines 12-14 define the *Label* referenced by the *Model's Prediction* using the keyword "labels". Finally, lines 16-18 define a *Deployment* instance named *callcentreDeployment*, with line 17 showing the *Model* that the *Deployment* references using the keyword "model".

4.2 Dataset Shift

This subsection introduces the metamodel classes used to specify the policies for detecting dataset shift and the subsequent application of corrective actions. These can be seen in Figure 4.2.

To detect dataset shift, data scientists implement algorithms using general-purpose languages and create *Algorithm* instances to represent them in a PDL model. For example, Listing 4.2 shows a Python [70] script that uses the Kolmogorov-Smirnov [71] statistical test to check if two datasets are sampled from the same underlying statistical distribution and lines 18-23 in

```
1 Model callcentreDecisionTree{
2   uses wait_duration, service_duration, is_solved
3   outputs callcentreDecisionTreePrediction
4   predicts is_happy
5 }
6
7 FeatureStore{
8   features
9     wait_duration,
10    service_duration,
11    is_solved
12  labels
13    is_happy
14 }
15
16 Deployment callcentreDeployment{
17   model callcentreDecisionTree
18 }
```

Listing 4.1: PDL model showcasing the metamodel's core classes.

Listing 4.3 show the corresponding *Algorihtm* instance. Notice how the script does not seem complete. Specifically, the script only defines a function that expects specific arguments and returns certain values based on the result of the statistical test. There is no code for fetching the relevant data and passing it to the function or any code that shows how the returned result is used.

Filling in the gaps mentioned above is the responsibility of algorithm runtimes. These are components of the ML platform implemented by software engineers capable of fetching data from an ML platform's data store, executing algorithms and handling the results of the execution. The developer of an algorithm runtime documents the requirements that the supported algorithm implementations must fulfil and makes them available to data scientists. In line 20 of Listing 4.3, for example, we can see that the "kolmogorovSmirnov" *Algorithm* references the "pythonFunction" *AlgorithmRuntime*. This runtime requires the algorithm to be implemented as a Python function with the input/output signature seen in Listing 4.2. Further technical details about algorithm runtimes are given in Chapter 5.

Given the attributes of the Kolmogorov-Smirnov algorithm, we can apply it to the call centre scenario to monitor the values of the "wait duration" variable against covariate shift. For this, the algorithm needs to receive as inputs the values of the "wait duration" variable in the training

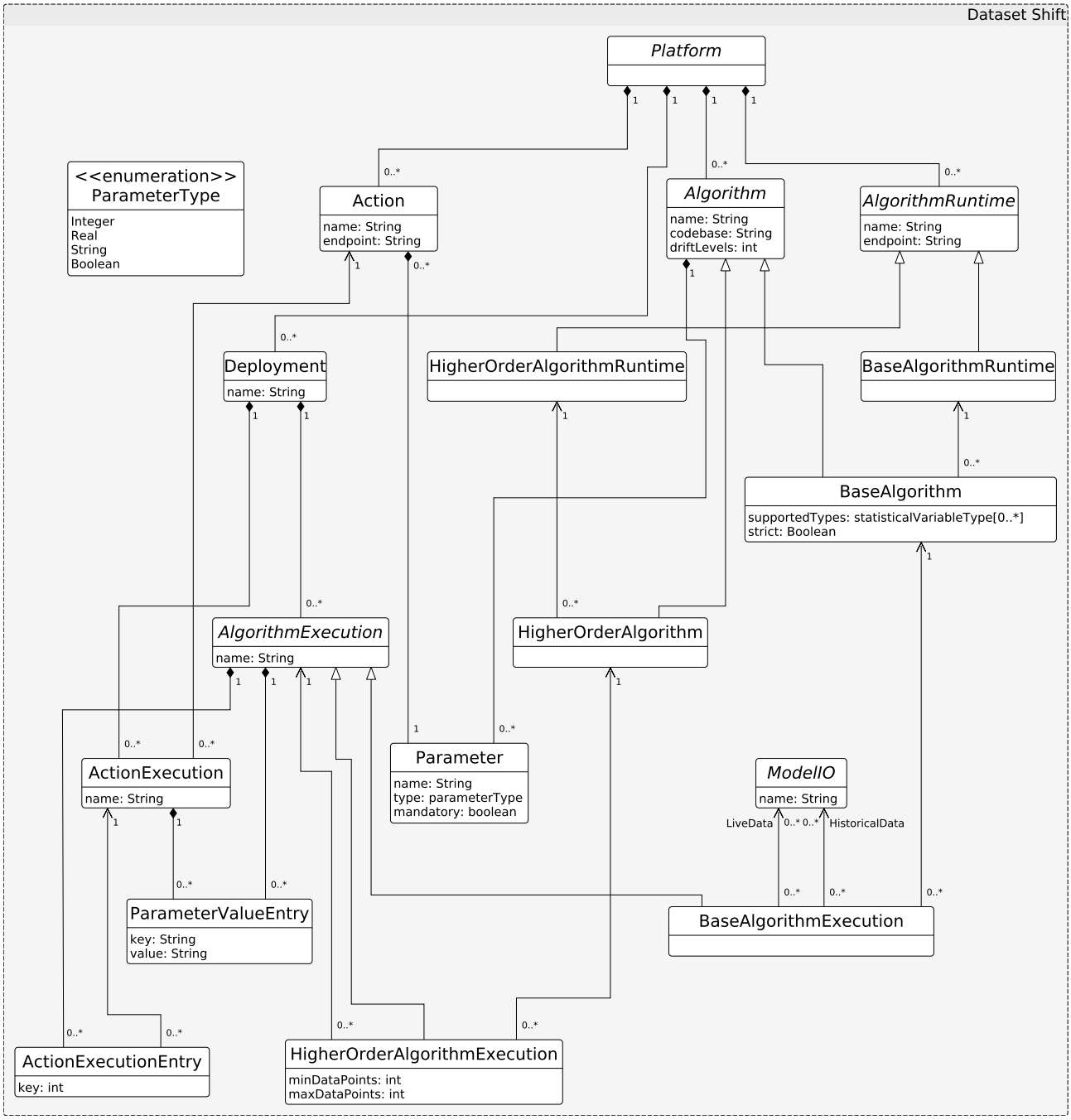


Figure 4.2: Classes of the PDL Metamodel related to Dataset Shift.

```
1 from scipy import stats
2
3 def ksTest(trainSet, liveSet, parameters):
4     pValue = stats.ks_2samp(trainSet, liveSet))[1]
5     if pValue < parameters['pValue']:
6         return 1, pValue
7     else:
8         return 0, pValue
```

Listing 4.2: Dataset shift detecting algorithm implementation in Python.

set and in recent prediction requests received by the deployed ML model. Information regarding the inputs of an algorithm to apply it in a specific scenario is contained in *AlgorithmExecution* instances, as seen for examples in lines 32-38 of Listing 4.3. Additionally, algorithms can be parameterised. The names of the parameters are defined in the *Algorithm* definition (e.g. line 22 of Listing 4.3), and their values are provided in the *AlgorithmExecution* (e.g. line 36 of Listing 4.3). The algorithm runtime passes the parameter values to the algorithm along with the rest of the inputs. Finally, *AlgorithmExecutions* also include a mapping that links the algorithm execution results to the relevant action execution (e.g. line 37 of Listing 4.3).

Related to the execution of actions in response to dataset shift are the *Action* and *ActionExecution* classes. An example of an action is sending an email notification to the data scientist that trained the affected ML model. Similarly to algorithm runtimes, action execution functionality is provided by specific software components in the underlying ML platform. These components are represented by *Action* instances in PDL models.

Based on the same principle that links *Algorithms* with *AlgorithmExecutions*, while *Action* instances represent a capability of the underlying platform, its usage is represented by an *ActionExecution* instance that parametrises the *Action* to fit in the context of a particular scenario. As an example, lines 40-43 of Listing 4.3 show the "emailMe" *ActionExecution* that uses the "email" *Action* defined in lines 25-27. The *ActionExecution* is parametrised in line 42 with the email address of the intended email recipient.

```
Model callcentreDecisionTree{
  uses wait_duration, service_duration, is_solved
  outputs callcentreDecisionTreePrediction
  predicts is_happy
}

FeatureStore{
  features
    wait_duration,
    service_duration,
    is_solved
  labels
    is_happy
}

BaseAlgorithmRuntime PythonFunction

BaseAlgorithm kolmogorovSmirnov{
  codebase "http://repo.com/kolmogorov-smirnov"
  runtime PythonFunction
  severity levels 2
  parameters pValue
}

Action email{
  parameters email
}

Deployment callcentre{
  model callcentreDecisionTree

  BaseAlgorithmExecution ksWaitTime{
    algorithm kolmogorovSmirnov
    live data wait_time
    historical data wait_time
    parameter values pValue = 0.05
    actions 1->emailMe
  }

  ActionExecution emailMe{
    action email
    parameter values email=user@company.com
  }
}
```

Listing 4.3: PDL model showing a simple dataset shift detection scenario.

4.2.1 Base and Higher Order Algorithms

Algorithm, *AlgorithmExecution* and *AlgorithmRuntime* are abstract classes. So far, the examples showcase the subclasses *BaseAlgorithm*, *BaseAlgorithmExecution* and *BaseAlgorithmRuntime*. Instances of *BaseAlgorithm* represent algorithms that can detect dataset shift from *ModelIO* values (i.e. features, predictions and labels) in the training and live datasets. On the other hand, *HigherOrderAlgorithm* instances represent algorithms that take as input a set of outputs from the execution of another algorithm.

As Listing 4.2 shows, the execution of a base algorithm returns a pair of values. The first value is an integer representing the presence and severity of dataset shift. It corresponds to the "keys" of the result-action execution map of the algorithm executions that utilise the algorithm. The possible values must be in the [0,N) range where N is the *severity levels* attribute of the relevant *Algorithm* instance. The second return value of the algorithm is the "raw result" and is meant to be used as input to higher-order algorithms for further analysis.

Listing 4.4 shows how a *HigherOrderAlgorithm* can be used. It expresses a scenario where a data scientist wants to increase covariate shift detection robustness by considering the last N execution results of the abovementioned Kolmogorov-Smirnov algorithm. An example algorithm that can be used for this is the exponential moving average that calculates a weighted average of the results, with more recent results having a higher weight.

From this listing, one can notice that a *HigherOrderAlgorithmExecution* has a few differences compared to a *BaseAlgorithmExecution*. One needs to define the *AlgorithmExecution* (Base or HigherOrder) whose output will serve as the input of the newly created *HigherOrderAlgorithmExecution* (seen in line 39 of Listing 4.4). In addition, the minimum and maximum number of observations must be defined (seen in lines 41-42 of Listing 4.4).

```
Model callcentreDecisionTree{
  uses wait_duration, service_duration, is_solved
  outputs callcentreDecisionTreePrediction
  predicts is_happy}

FeatureStore{
  features wait_duration, service_duration, is_solved
  labels is_happy}

BaseAlgorithmRuntime PythonFunction

BaseAlgorithm kolmogorovSmirnov{
  codebase "http://repo.com/kolmogorov-smirnov"
  runtime PythonFunction
  severity levels 2
  parameters pValue}

HigherOrderAlgorithmRuntime higherOrderPythonFunction

HigherOrderAlgorithm exponential-moving-average{
  codebase "http://repo.com/exponential-moving-average"
  runtime higherOrderPythonFunction
  parameters alpha,threshold
  severity levels 2}

Action email{
  parameters email}

Deployment callcentre{
  model callcentreDecisionTree

  BaseAlgorithmExecution ksWaitTime{
    algorithm kolmogorovSmirnov
    live data wait_time
    historical data wait_time
    parameter values pValue = 0.05}

  HigherOrderAlgorithmExecution ema-service-duration{
    algorithm exponential-moving-average
    observed execution ksWaitTime
    min observations 3
    max  observations 3
    parameter values alpha = 0.5, threshold = 0.05
    actions 1->emailMe}

  ActionExecution emailMe{
    action email
    parameter values email=user@company.com}
}
```

Listing 4.4: PDL model showing the usage of higher order algorithms.

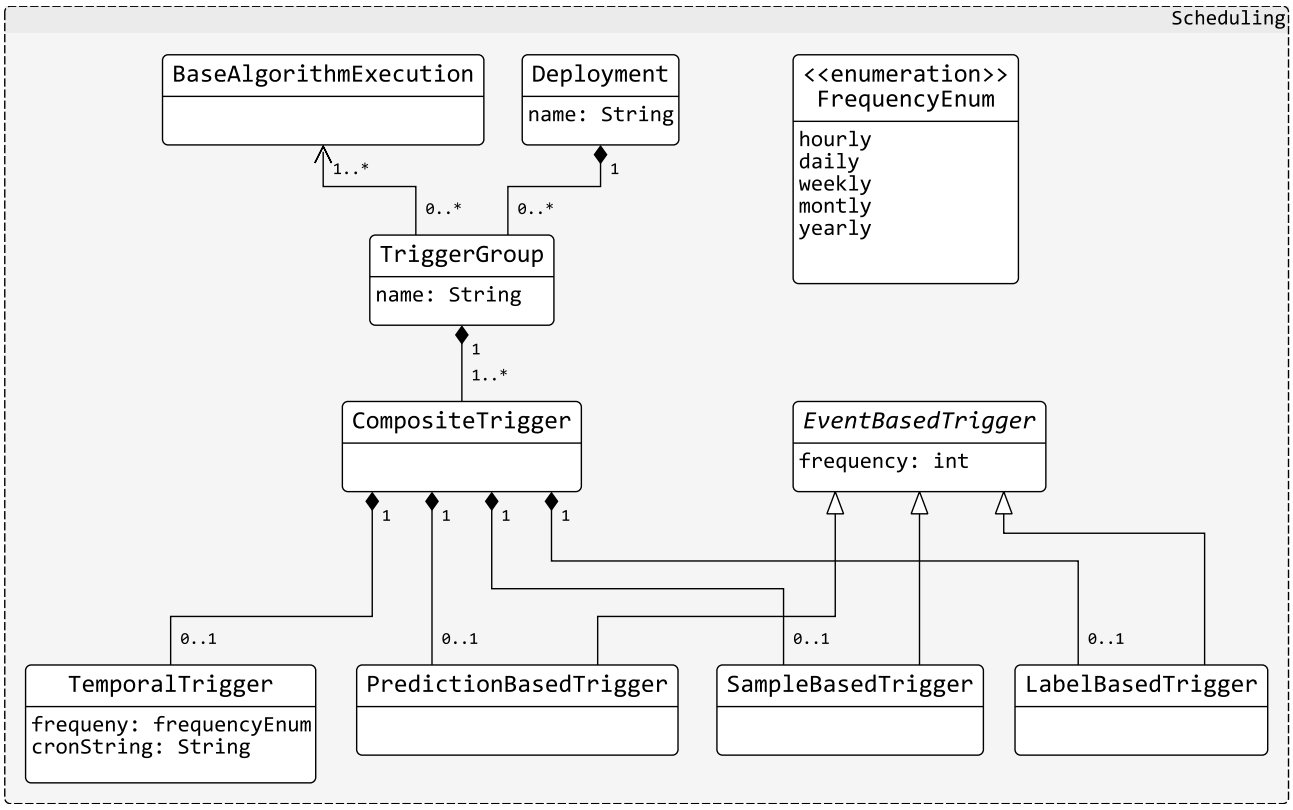


Figure 4.3: Classes of the PDL metamodel related to scheduling.

4.3 Scheduling

An essential aspect of the modelled domain is defining the frequency of algorithm executions and the subsequent execution of corrective actions as needed. For this purpose, classes that represent various triggers and classes that represent a combination of triggers are provided. More specifically, *TemporalTrigger*, *SampleBasedTrigger*, *PredictionBasedTrigger*, and *LabelBasedTrigger* can express the frequency of one or more algorithm executions in terms of how long it has been since the latest execution, how many unlabelled samples have been received, how many predictions have been served and how many labels have been received for previously unlabelled samples respectively. For more complex scenarios, data scientists can create *CompositeTrigger* instances containing up to one instance for each kind of individual trigger and represent scenarios in which the requirements must be met for multiple individual triggers simultaneously. Lastly, *CompositeTrigger* instances can be grouped in a *TriggerGroup* instance. A *TriggerGroup* instance tracks which *BaseAlgorithmExecution* is to be executed when the requirements for at least one of the contained *TriggerGroups* are met.

Lines 51-53 of Listing 4.5 shows a trigger group has been added to the "callcentre" *Deployment* to showcase all possible combinations. It should be noted that trigger groups only trigger base algorithm executions as higher order algorithm executions are triggered indirectly when the algorithm execution they observe executes (provided that the observed algorithm execution has been triggered at least the minimum number of times required).

4.4 Validation

Although the metamodel of PDL already captures certain typing and cardinality constraints, there is still the possibility of creating a model that contains errors according to the semantics of the domain. This Section discusses a series of constraints, based on the semantics of the domain, that are used to validate PDL models. This section also explains certain parts of the metamodel that have yet to be presented. These are not necessary to describe dataset shift strategies, but including them in a PDL model enables the error detection features.

Listing 4.6 shows an example PDL model that uses features of the DSL that enable the optional feature and type validation.

4.4.1 Feature Validation

Feature validation ensures that every deployment uses a suitable ML model. This is true when every feature used as input by the ML model is retrievable from the deployment's inputs. This validation can prove valuable when a deployment transitions to a newer ML model utilising more or different features to ensure compatibility with the existing prediction consumers.

For instance, in the call centre scenario, for a specific call, one can retrieve the associated "wait_duration", "service_duration", and "is_solved" values. Thus, instead of sending the actual values of the variables, a prediction requester can send just a "call ID" that can be used to uniquely identify the call and retrieve all the necessary inputs for a customer satisfaction prediction. If additional features are available per call, these can be added as inputs to a

```
Model callcentreDecisionTree{
  uses wait_duration, service_duration, is_solved
  outputs callcentreDecisionTreePrediction
  predicts is_happy}

FeatureStore{
  features wait_duration, service_duration, is_solved
  labels is_happy}

BaseAlgorithmRuntime PythonFunction

BaseAlgorithm kolmogorovSmirnov{
  codebase "http://repo.com/kolmogorov-smirnov"
  runtime PythonFunction
  severity levels 2
  parameters pValue}

HigherOrderAlgorithmRuntime higherOrderPythonFunction

HigherOrderAlgorithm exponential-moving-average{
  codebase "http://repo.com/exponential-moving-average"
  runtime higherOrderPythonFunction
  parameters alpha, threshold
  severity levels 2}

Action email{
  parameters email}

Deployment callcentre{
  model callcentreDecisionTree

  BaseAlgorithmExecution ksWaitTime{
    algorithm kolmogorovSmirnov
    live data wait_time
    historical data wait_time
    parameter values pValue = 0.05}

  HigherOrderAlgorithmExecution emaWaitTime{
    algorithm exponential-moving-average
    observed execution ksWaitTime
    min observations 3
    max  observations 3
    parameter values alpha = 0.5, threshold = 0.05
    actions 1->emailMe}

  ActionExecution emailMe{
    action email
    parameter values email=user@company.com}

  Trigger t1{
    every 100 samples 100 predictions 100 labels
    or every one day
    execute ksWaitTime}
}
```

Listing 4.5: PDL model showing a dataset shift detection scenario with scheduling.

```
FeatureStore{
  entities
    call{keys callID}
  request data additional_data
  features
    wait_duration:continuous{requires entities call},
    service_duration:continuous{requires entities call},
    is_solved:categorical{requires entities call},
    additional_feature{requires request data additional_data}
  labels
    is_happy:categorical
}

Model callcentreDecisionTree{
  uses wait_duration, service_duration, is_solved
  outputs happiness_prediction
  predicts is_happy
}

BaseAlgorithmRuntime PythonFunction

BaseAlgorithm kolmogorovSmirnov{
  codebase "http://repo.com/kolmogorovsmirnov"
  runtime PythonFunction
  severity levels 2
  accepts only continuous
  parameters mandatory pValue:Real
}

Action email{
  parameters mandatory email:String
}

Deployment callcentre{
  inputs callID
  model callcentreDecisionTree

  BaseAlgorithmExecution ksWaitTime {
    algorithm kolmogorovSmirnov
    live data service_duration
    historical data service_duration
    actions 1->emailMe
    parameter values pValue = 0.05
}

ActionExecution emailMe{
  action email
  parameter values email=user@company.com
}

  Trigger t1{
    every 100 samples
    execute ksWaitTime
  }
}
```

Listing 4.6: PDL model utilising validation features.

newer version of the satisfaction prediction ML model while keeping the data that prediction consumers have to send unchanged.

This is reflected in the PDL metamodel with a pattern already familiar to data scientists using a feature store in their workflow. Every *Feature* can reference zero or more *Entities*. An *Entity* represents a concept in the domain within which a data scientist builds a predictive model, for example, a call in the call centre scenario. *Entities* contain one or more *Keys* which can uniquely identify them. *Keys* can also be referenced by a *Deployment* as inputs. For a PDL model to be valid, every *Deployment* must reference the relevant *Keys* such that all features that the *Deployment's* referenced *Model* uses can be retrievable.

Alternatively, some features are calculated on-the-fly based on data only available when a prediction is requested. This, for example, could be a search query that a user submits to a search engine. This is represented in the PDL metamodel by the *RequestData* class. If a *Feature* references a *RequestData* instance, this must also be referenced as input by any *Deployment* that uses an ML *Model* that uses said *Feature*.

Algorithm 1 shows in pseudocode the validation steps performed for feature validation.

Algorithm 1 Feature Validation

```
function checkDeploymentInputs(deployment)
    deploymentInputs ← deployment.getInputs()
    mlModel ← deployment.getMLModel()
    features ← mlModel.getInputs()
    for each feature in features do
        entities ← feature.getEntities()
        for each entity in entities do
            keys ← entity.getKeys()
            for each key in keys do
                if key not in deploymentInputs then
                    warning()
        requestData ← feature.getRequestData()
        for each requestDatum in RequestData do
            if requestDatum not in deploymentInputs then
                warning()
```

4.4.2 Parameter validation

Instances of *Parameter* are contained by *Algorithm* and *Action* instances to denote that they are parameterisable at runtime. An additional model validation procedure that can be performed with regard to *Parameters* consists of checking that the *ParameterValueMap* contained in *AlgorithmExecution* and *ActionExecution* instances have valid keys-value entries based on the *Algorithm* or *Action* that the *Execution* references. For example, in Listing 4.6, the *kolmogorovSmirnov BaseAlgorithm* contains a *Parameter* named "pValue" of type "Real". Therefore it should be checked that every *AlgorithmExecution* that references this *BaseAlgorithm* contains a *ParameterValueEntry* with "pValue" as the key and a value that is a valid *real* number. Additionally, we check that every *Parameter* denoted as mandatory is given a value.

Algorithm 2 shows in pseudocode the steps performed to verify that every mandatory parameter is given a value and that those values are valid according to the expected data types.

4.4.3 Statistical Type Validation

The last type of validation checks that every *BaseAlgorithmExecution* takes in as input valid *ModelIO* based on the selected *BaseAlgorithm*. To enable this check, the *type* attribute of *Feature* and *Label* instances can optionally be given a value. This *type* attribute is enumerative. It can take the values *continuous*, *categorical* and *orderedCategorical* that correspond to the different types of statistical variables found in the literature [21]. Additionally, *BaseAlgorithms* can optionally be given one or more values of the same enumerative type in their *supportedTypes* attribute. Based on this information, we can validate whether the *ModelIO* instances referenced by a *BaseAlgorithmExecution* are of suitable statistical type for the *BaseAlgorithm* used. Additionally, the *strict* attribute of a *BaseAlgorithm* is can be set. Depending on its value, only a warning will be generated in case of statistical variable type mismatch instead of an error, which will not cause the rejection of the validated PDL model but still notify the user of the potential problem.

For example, in line 26 of Listing 4.6, the keywords "accepts only" specify that the *kolmogorovS-*

Algorithm 2 Parameter Validation

```
function checkMandatory(execution)
    algorithm ← execution.getAlgorithm()
    parameters ← algorithm.getAdditionalParameters()
    parameterValueMap ← execution.getParameterValueMap()
    for each parameter in parameters do
        if parameter.isMandatory() then
            for each entry in parameterValueMap do
                if parameter not in parameterValueMap then
                    error()
function checkDataTypes(execution)
    algorithm ← execution.getAlgorithm()
    parameters ← algorithm.getAdditionalParameters()
    parameterValueMap ← execution.getParameterValueMap()
    for each entry in parameterValueMap do
        key ← entry.getKey()
        value ← entry.getValue()
        for each parameter in parameters do
            if key = parameter.getName() then
                parameterType ← parameter.getType()
                break;
        if parameterType = "Integer" then
            checkValidInteger(value)
        else if parameterType = "Boolean" then
            checkValidBoolean(value)
        else if parameterType = "Real" then
            checkValidReal(value)
```

mirnov BaseAlgorithm only supports *continuous* statistical variables as inputs. This is because while variants of the Kolmogorov-Smirnov statistical test can be used for statistical variables of ordered categorical type [72], the algorithm implementation shown in Listing 4.2 depends on the Kolmogorov-Smirnov implementation found in the SciPy Python package [73] which only supports continuous statistical variables. This could lead to errors that are difficult to detect since the data scientist that implemented the algorithm could be other than the one using the algorithm to detect covariate shift in a specific context. To avoid this situation, this type of validation is provided.

Algorithm 3 shows in pseudocode the steps that are taken to verify that the *currentIOValues* and *historicalIOValues* that are referenced by *BaseAlgorithmExecutions* have valid statistical types for the selected *BaseAlgorithm*.

Algorithm 3 Statistical Type Validation

```
function checkStatisticalTypes(execution)
    algorithm ← execution.getAlgorithm()
    supportedTypes ← algorithm.getSupportedTypes()
    strict ← algorithm.isStrict()
    currentIOValues ← execution.getCurrentIOValues()
    for each io in currrentIOValues do
        statisticalType ← io.getType()
        if statisticalType not in supportedTypes then
            if strict then
                error()
            else
                warning()
    historicalIOValues ← execution.getHistoricalIOValues()
    for each io in historicalIOValues do
        statisticalType ← io.getType()
        if statisticalType not in supportedTypes then
            if strict then
                error()
            else
                warning()
```

4.5 Editing Support

To provide data scientists with a simple user experience for creating PDL models and submiting them to the orchestrator, an Xtext-based web editor was developed. As shown in figure 4.4, the editor offers various convenient features such as syntax highlighting, syntax error detection and auto-completion.

The syntax error detection, which is automatically generated from the Xtext grammar file, does not cover the kind of semantic validation described in the previous section. To cover this, a custom validation class that implements the algorithms presented in the previous section was created. In this manner, the editor backend service continuously performs model validation and users are given immediate feedback while editing.

Another nice feature of a web-based editor is that it does not require installation. When the user has finished creating or updating a PDL model, the editor backend service parses the textual syntax and generates an equivalent model in XMI. The XMI document is then sent to the orchestrator to be processed. Following this pattern, the orchestrator only needs to support XMI, while multiple concrete syntaxes can be developed based on users' preferences.

4.6 Chapter Summary

This chapter introduced the Panoptes Description Language (PDL), which is designed to facilitate the deployment of ML monitoring workflows. The chapter provided an overview of the PDL metamodel and textual syntax that can be used to specify the different aspects of a monitoring workflow. The chapter also documents the model validation features of PDL that aid data scientists in the specification of error-free PDL models. The next chapter focuses on the technical aspects of Panoptes, which have only been lightly covered in this chapter.

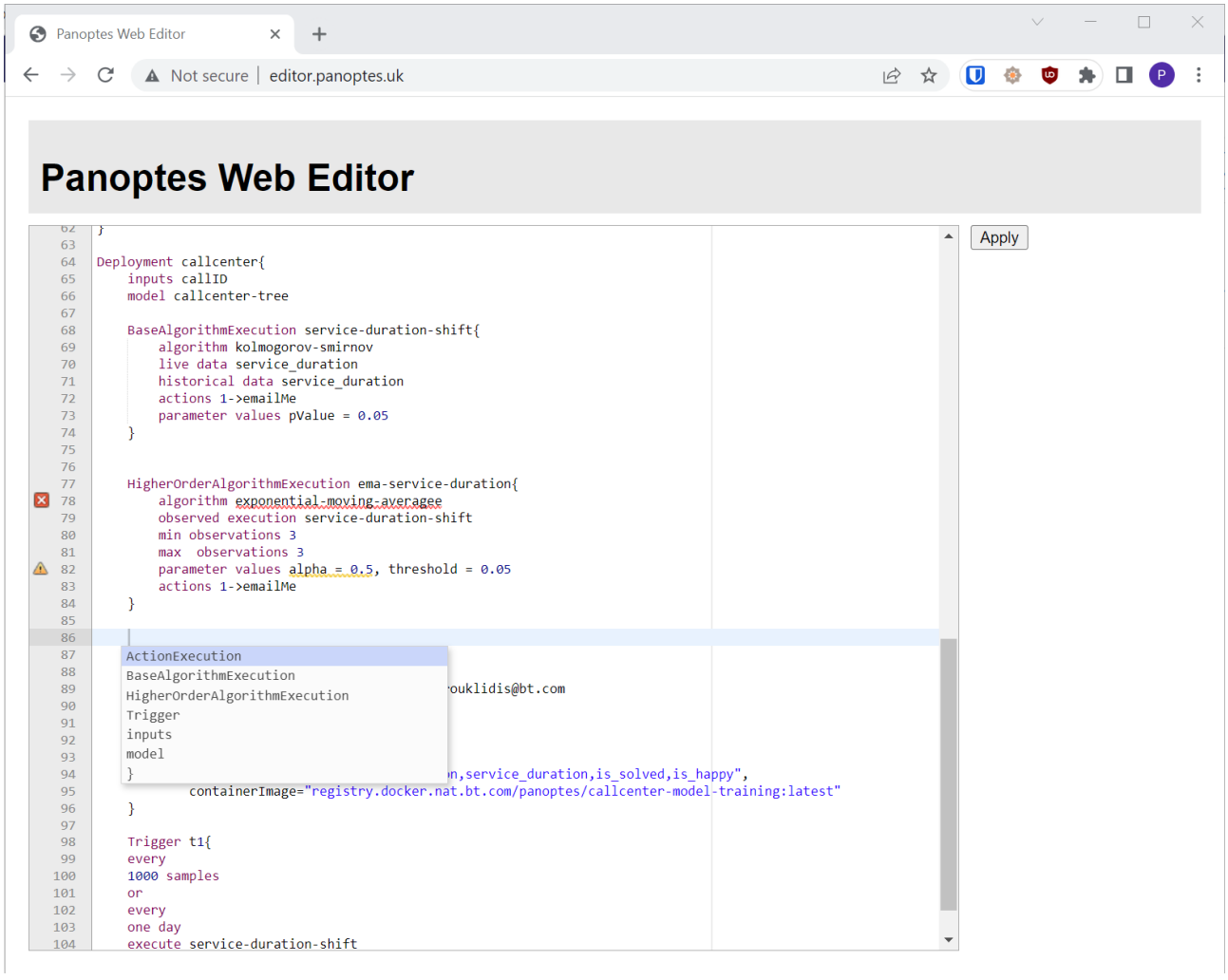


Figure 4.4: The Panoptes web editor user interface.

Chapter 5

Monitoring of ML Systems

This chapter covers the technical decisions that must be made to produce a concrete implementation of Panoptes from the conceptual description presented in Chapters 3 and 4, which leaves certain technical decisions undefined since these can vary between implementations.

The rest of the chapter is structured as follows: Section 5.1 covers the infrastructure used as a base to develop the reference implementation. Section 5.2 covers the serialisation format of the different events exchanged between the various components. The rest of the chapters' sections then cover the technical details of all the major components of a Panoptes implementation as well as the process that can be followed to adopt Panoptes.

5.1 Base Infrastructure

Chapter 3 described the Panoptes architecture as service-oriented and event-driven. However, to satisfy the requirements of these high-level architectural styles, there are various practical decisions that an implementer has to make. The reference implementation uses modern, cloud-native technologies to implement Panoptes as a set of loosely coupled containerised web services.

In the interest of making Panoptes easier for organisations to adopt, the usage of well-established technologies was deemed important. All the technologies described in this section are industry-

standard, open-source technologies, hosted by the Cloud-Native Computing Foundation (CNCF). The CNCF is a non-profit organisation that in addition to hosting the development of cloud-related open source projects, offers additional services such as training and certification programs.

5.1.1 Kubernetes

Kubernetes is an open-source container orchestration platform designed to automate the deployment, scaling, and management of containerised applications. It was initially developed by Google but has been donated to CNCF, which oversees its development and keeps the project vendor-neutral. Kubernetes has become a cornerstone in the world of cloud-native applications, offering a robust and extensible framework for managing complex systems at scale.

Central to Kubernetes is the concept of containers. Containers are lightweight, standalone, and executable software packages encompassing an application and all its dependencies, ensuring consistent environments across different development and deployment stages. This consistency is particularly valuable in service-oriented architectures, where an application is decomposed into small, independent services that run in their own containers.

Kubernetes plays a pivotal role in the implementation and management of service-oriented architectures by providing the following valuable features:

Service Discovery and Load Balancing: In a service-oriented architecture, services need to discover and communicate with each other. Kubernetes provides built-in service discovery, allowing services to find each other by name and load balancing to distribute traffic among multiple service instances.

Automatic Scaling: As demand for a particular service increases, Kubernetes can automatically scale the number of container instances, ensuring the system can handle varying loads without manual intervention.

Self-healing: If a container or a node fails, Kubernetes can detect the failure and restart it or reschedule it to a healthy node, ensuring high availability and resilience.

Rollouts and Rollbacks: Kubernetes facilitates continuous deployment and integration. New versions of services can be rolled out gradually, and if issues arise, rolled back to a previous stable version.

Configuration and Secret Management: In a service-oriented architecture, services often require configuration data or secrets (like API keys). Kubernetes provides mechanisms to manage and inject these configurations and secrets, ensuring security and consistency.

Resource Management: Kubernetes allows for setting CPU and memory quotas for containers, ensuring that each service gets the resources it needs and that no single service can monopolise shared resources.

In conclusion, Kubernetes, with its rich set of features tailored for container management, has emerged as an indispensable tool for implementing service-oriented architectures. By providing automated deployment, scaling, and management capabilities, Kubernetes ensures that SOA principles can be realised with efficiency, resilience, and scalability in the modern cloud-native landscape.

5.1.2 KNative

KNative is an open-source platform that extends Kubernetes to provide a set of middleware components for serverless applications. It aims to simplify the complexities associated with building, deploying, and running cloud-native applications, allowing developers to focus on writing code without the intricacies of the underlying infrastructure. KNative includes two main components:

Serving: Provides a rapid deployment model with features like automatic scaling, rollouts, and rollbacks. It ensures that applications can scale to zero when not in use, optimising resource utilisation.

Eventing: Facilitates the creation of event-driven applications by offering primitives to consume and produce events. It supports various event sources and allows for flexible event routing.

For the implementation of the event-driven aspects of the Panoptes architecture, KNative's eventing component was instrumental by providing the following functionality:

Event Sources: KNative supports a plethora of event sources, from traditional ones like databases and message queues to cloud-native sources such as cloud storage and analytics platforms. This flexibility allows systems to react to a diverse set of stimuli.

Event Routing: Once an event is detected, it needs to be routed to the appropriate service or function. KNative's eventing component provides mechanisms to filter and route these events based on their attributes, ensuring they reach the right destination.

In conclusion, KNative, with its robust serving and eventing components, has emerged as a cornerstone in the development of event-driven architectures in the cloud-native ecosystem. By abstracting the complexities of infrastructure management and providing tools to seamlessly integrate events into applications, KNative empowers developers to build responsive, scalable, and resilient systems.

5.2 Event Serialization Format

Another practical decision that needed to be made for the reference implementation of Panoptes was the serialisation format of the events that the various services need to produce and consume. All serialised events follow the CloudEvents v1.0 specification, the format supported by KNative eventing.

The CloudEvents specification is designed to help ease event declaration and delivery across services. The specification details how an event should be formatted. An event generally has a number of attributes which can be used to describe the event. An event has the following attributes:

Data: Contains the event payload. The schema of this is usually application-specific.

Data Content Type: Describes the data content type. For example, application/json.

Id: A unique identifier for the event.

Source: Identifies the context in which an event happened, usually a URI.

Specversion: The version of the CloudEvents specification which the event uses.

Type: Describes the type of event related to the originating occurrence.

Time: Timestamp of when the event happened.

Subject: Describes the subject of the event in the context of the event producer.

Extensions: Additional metadata without a prescribed meaning. These are named using lowercase alphabetic characters.

The CloudEvents spec does not mandate a transport protocol but defines several bindings, including HTTP, AMQP, MQTT, Kafka, and NATS. These bindings help ensure consistency when transmitting CloudEvents over different protocols.

Additionally, an event can be serialised using one of two content modes. In structured content mode, the event is serialised into a document, and the document is transported as the transport

protocol's message body. In binary content mode, the event attributes are mapped to message protocol headers/metadata, and the event data is transported as the message body without modification.

For the Panoptes implementation, events were serialised in structured content mode and transported over HTTP in the Json data format. In the following sections that discuss the technical details of the various Panoptes components, the different messages that each component produces will also be given.

5.3 ML Platform Implementation

As discussed in Chapter 2, an ML platform comprises multiple components supporting the stages of the typical ML workflow. The ML platform is a component that exists independently of Panoptes but still needs to produce the events that the orchestrator expects. As a proof of concept, a minimal ML platform has been implemented to show how the interfacing of Panoptes deployments with pre-existing ML platforms can be approached. This ML platform was also utilised to conduct the empirical studies presented in Chapter 6.

The implementation of the example ML platform was based on Kubeflow. Kubeflow is a platform optimised for ML operations within a Kubernetes environment. It offers tools for various stages of the ML lifecycle, from data preprocessing to model deployment. Kubeflow's integration simplifies the often complex process of managing ML workflows, making them more accessible and scalable.

The component of Kubeflow most relevant for the minimal ML platform implemented to help showcase Panoptes' features is KServe, which is the component that offers ML model deployment functionality. A KServe inference service can produce events in the CloudEvent v1.0 format for every request it receives, making it easy to integrate with Panoptes.

Listings 5.1 and 5.2 show examples of the two kinds of events a KServe inference service can produce. Listing 5.1 is an event produced when a request is received. On the other hand,

```
{
    "specversion" : "1.0",
    "type" : "org.kubeflow.serving.inference.request",
    "source" : <The URL of the Model Server>,
    "subject": null,
    "id" : <Random Identifier>,
    "time" : <Time that the Event was emitted>,
    "datacontenttype" : "application/json",
    "data" : {
    <The feature values used to produce the prediction>
    }
}
```

Listing 5.1: KServe Request Logging Event.

```
{
    "specversion" : "1.0",
    "type" : "org.kubeflow.serving.inference.request",
    "source" : <The URL of the Model Server>,
    "subject": null,
    "id" : <Random Identifier>,
    "time" : <Time that the Event was emitted>,
    "datacontenttype" : "application/json",
    "data" : {
    <The Model's prediction>
    }
}
```

Listing 5.2: KServe Response Logging Event.

Listing 5.2 shows an event produced when a response is sent back.

As it can be seen in the listings, the events follow While the above events contain information relevant to the orchestrator, they are not in a format that it supports. For this reason, the events are first consumed by an event transformation service that receives the information and produces equivalent events in the format that the orchestrator expects. Listings 5.3 and 5.4 show the events produced by the event transformation service.

In addition to the above, the ML platform is responsible for emitting one additional event

```
{
    "specversion" : "1.0",
    "type" : "org.lowcomote.panoptes.trigger.sample",
    "source" : "eventTransformer",
    "subject": null,
    "id" : <Random identifier>,
    "time" : <Time that the event was emitted>,
    "datacontenttype" : "application/json",
    "data" : {
        "triggerType": "sample",
        "count": <The number of samples in the request in case of batch prediction>,
        "deployment": <The name of the deployment that this event is for>
    }
}
```

Listing 5.3: Panoptes Sample Trigger Event.

```
{
    "specversion" : "1.0",
    "type" : "org.lowcomote.panoptes.trigger.prediction",
    "source" : "<The URL of the Model Server>",
    "subject": null,
    "id" : "<Random Identifier>",
    "time" : "<Time that the Event was emitted>",
    "datacontenttype" : "application/json",
    "data" : {
        "triggerType": "prediction",
        "count": <The number of predictions in the response in case of batch prediction>,
        "deployment": <The name of the deployment that this event is for>
    }
}
```

Listing 5.4: Panoptes Prediction Trigger Event.

```
{
    "specversion" : "1.0",
    "type" : "org.lowcomote.panoptes.trigger.prediction",
    "source" : "<The URL of the Model Server>",
    "subject": null,
    "id" : "<Random Identifier>",
    "time" : "<Time that the Event was emitted>",
    "datacontenttype" : "application/json",
    "data" : {
        "triggerType": "label",
        "count": <The number of ground truth labels ingested>,
        "deployment": <The name of the deployment that this event is for>
    }
}
```

Listing 5.5: Panoptes Label Trigger Event.

whenever a ground truth label is ingested. Listing 5.5 shows the format of these events.

5.4 Panoptes Orchestrator

As discussed in Chapter 3, the orchestrator ingests the PDL models received from the model editor and is responsible for implementing the runtime behaviour specified in them. While the PDL metamodel is designed so that data scientists can conveniently specify how deployed ML models are to be monitored, it does not provide the orchestrator with a way to keep track of its internal state to determine when algorithm and action executions must be triggered. For this reason, after the orchestrator receives a PDL model, it constructs Finite State Machine (FSM) objects that directly map to the required runtime behaviour.

Specifically, an FSM object is constructed for every *Deployment* in a processed PDL model. As shown in Figure 5.1, this FSM only has one state, labelled *STANDBY*, and multiple transitions

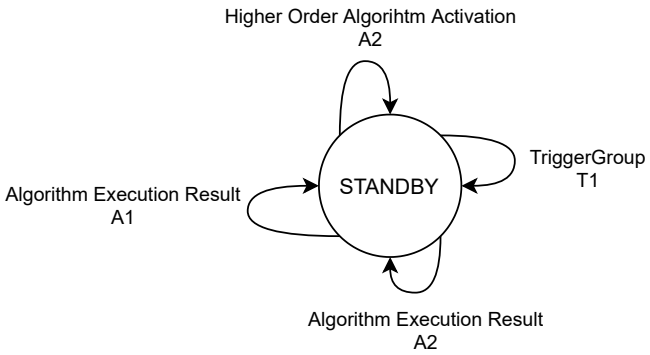


Figure 5.1: Example FSM representation of a PDL model used by the orchestrator.

from *STANDBY* back to itself that are triggered based on the events ingested by the FSM. Whenever a transition is triggered, an accompanying function is executed that sends out an event message to either an algorithm runtime or action service as needed. Transitions are added to the FSM according to the following rules:

- One transition is added for every *TriggerGroup* a *Deployment* contains. These trigger whenever a base algorithm must be executed and will always send an event message to the relevant algorithm runtime.
- One transition is added for every *AlgorithmExecution* a *Deployment* contains. These trigger after an algorithm execution finishes. Based on the result of the execution, if an action needs to be executed, a message is sent out to the relevant action service.
- One additional transition is added if the above *AlgorithmExecution* is a *HigherOrderAlgorihtmExecution*. These trigger whenever the observed algorithm executes. If the observed algorithm has been executed a sufficient number of times, a message is sent out to the algorithm runtime of the higher-order algorithm.

For the transitions to be triggered at the right time, the ML platform need to sends the events shown in the previous section message to the orchestrator whenever a monitoring-related event occurs. These events correspond to the *EventBasedTrigger* classes. No external messages are needed for *TemporalTriggers* since the orchestrator can independently keep track of the amount of time passed since the last time a *TriggerGroup* transition was triggered. The events corresponding to *SampleBasedTriggers* and *PredictionBasedTriggers* are emitted by a model

```
{
    "specversion" : "1.0",
    "type" : "org.lowcomote.panoptes.higherOrderAlgorithmExecution.trigger",
    "source" : "<The URL of the Model Server>",
    "subject": <Name of the receiving runtime>,
    "id" : "<Random Identifier>",
    "time" : "<Time that the Event was emitted>",
    "datacontenttype" : "application/json",
    "data" : {
        "name": <Name of the algorithm>,
  "codebase": <Repository containing the algorithm's source code>
    }
}
```

Listing 5.6: Algorithm Creation Request Event.

server when it responds to a prediction request. These inform the orchestrator that a new sample has been received and was used to produce a new prediction. Events corresponding to *LabelBasedTriggers* are emitted by any service that ingests a ground-truth label for a previously served prediction. Once enough messages have been accumulated to satisfy the *TriggerGroup*conditions, the transition occurs, and the corresponding counters are reset.

5.5 Algorithm Runtime Implementation

As mentioned in Chapter 4, algorithm runtime services are responsible for packaging the algorithms that data scientists develop and executing them with the right inputs when asked by the orchestrator.

For the packaging step, an algorithm runtime service must be able to ingest events informing it of the creation of a new algorithm. The orchestrator sends this message whenever a PDL model introduces a new algorithm. The event message includes the algorithm's git repository from which the runtime can retrieve all the artefacts needed for the packaging. In the Python function runtime, for example, when an algorithm creation message is received, the runtime clones the repository and copies the file containing the algorithm function and the pip [74] requirements file into a Python project template. From that, a docker image is built that, when executed, can fetch the relevant data, execute the algorithm, and send a message to the orchestrator with the execution result. Listing 5.6 shows an example of an algorithm creation event.

```
{
    "specversion" : "1.0",
    "type" : "org.lowcomote.panoptes.baseAlgorithmExecution.trigger",
    "source" : "<The URL of the Model Server>",
    "subject": <Name of the receiving runtime>,
    "id" : "<Random Identifier>",
    "time" : "<Time that the Event was emitted>",
    "datacontenttype" : "application/json",
    "data" : {
        "modelName": <Name of the monitored ML model>,
        "deploymentName": <Name of the deployment>,
        "historicalFeatures": <The names of the historical values to be used in the algorithm execution>,
        "liveFeatures": <The names of the recent values to be used in the algorithm execution>,
        "baseAlgorithmExecutionName": <The name of the algorithm execution>,
        "startDate": <Starting time for retrieval of recent data>,
        "endDate": <Ending time for retrieval of recent data>,
        "algorithmName": <The name of the algorithm to be executed>,
        "parameters": {<Parameter names and coresponding value pairs>}
    }
}
```

Listing 5.7: Base Algorithm Execution Request Event.

For the execution step, an algorithm runtime service must be able to ingest a message instructing it to execute a previously packaged algorithm with specific inputs. The messages received for this step differ slightly between base and higher-order algorithm runtimes. For base algorithm runtimes, the message includes the names of the features/predictions/labels to be used as inputs. Listing 5.7 shows an example of a base algorithm execution event. After receiving it, the runtime triggers the execution of the packaged algorithm. It passes the names in as environment variables so the actual values can be fetched from a platform-specific data repository.

On the other hand, for higher-order runtimes, the message will include the name of the observed algorithm execution instead. Listing 5.8 shows an example of such an event. After receiving it, the runtime will similarly trigger the execution of the packaged algorithm with the name passed in as an environment variable. The past execution results of the observed algorithm execution will be retrieved by an API that the orchestrator exposes.

Once the execution of either type of algorithm execution is complete, the runtimes emmit an event that contains the result of the execution so that the orchestrator can request an action execution if necessary. Listing 5.9 shows an example of such an event

```
{
    "specversion" : "1.0",
    "type" : "org.lowcomote.panoptes.higherOrderAlgorithmExecution.trigger",
    "source" : "<The URL of the Model Server>",
    "subject": <Name of the receiving runtime>,
    "id" : "<Random Identifier>",
    "time" : "<Time that the Event was emitted>",
    "datacontenttype" : "application/json",
    "data" : {
        "deploymentName": <Name of the deployment>,
        "startDate": <Starting time of the period>,
        "endDate": <Ending time of the period>,
        "higherOrderAlgorithmName": <The name the higher order algorihtm to be executed>,
        "higherOrderalgorithmExecutionName": <The name of the algorithm execution>,
        "observedAlgorithmExecutionName": <The name of the observed algorihtm execution>,
        "windowSize": <How many results of the observed algorithm execution are needed>,
        "parameters": {<Parameter names and coresponding value pairs>}
    }
}
```

Listing 5.8: Higher Order Algorithm Execution Request Event.

```
{
    "specversion" : "1.0",
    "type" : <"org.lowcomote.panoptes.baseAlgorithmExecution.result" or
    org.lowcomote.panoptes.higherOrderAlgorithmExecution.result>,
    "source" : "<The URL of the Model Server>",
    "subject": <Name of the receiving runtime>,
    "id" : "<Random Identifier>",
    "time" : "<Time that the Event was emitted>",
    "datacontenttype" : "application/json",
    "data" : {
        "deployment": <Name of the deployment>,
        "startDate": <Starting time of the period>,
        "endDate": <Ending time of the period>,
        "algorithmExecution": <The name of the algorithm execution>,
        "level": <The discretised result of the algorithm execution that caused the action
    execution>,
        "rawResult": <The raw result of the algorithm execution that caused the action
    execution>

    }
}
```

Listing 5.9: Algorithm Execution Result Event.

```
{
    "specversion" : "1.0",
    "type" : "org.lowcomote.panoptes.actionExecution.trigger",
    "source" : "<The URL of the Model Server>",
    "subject": <Name of the receiving runtime>,
    "id" : "<Random Identifier>",
    "time" : "<Time that the Event was emitted>",
    "datacontenttype" : "application/json",
    "data" : {
        "deployment": <Name of the deployment>,
        "startDate": <Starting time of the period>,
        "endDate": <Ending time of the period>,
        "algorithmExecution": <The name of the algorithm execution that caused the action
    execution>,
        "parameters": {<Parameter names and coresponding value pairs>},
        "level": <The discretised result of the algorithm execution that caused the action
    execution>,
  "rawResult": <The raw result of the algorithm execution that caused the action execution>
    }
}
```

Listing 5.10: Action Execution Request Event.

5.6 Action Implementation

Action services are more straightforward to implement than algorithm runtime services since they do not have to execute any code dynamically. The only message that an action service must be able to ingest is instructing it to perform the action it represents. The message includes values of the parameters that the action accepts and auxiliary information, such as the name of the algorithm execution that has caused the action to execute in case it is useful. Listing 5.10 shows an example of such an event.

5.7 Adoption Process

This section presents the steps that need to be followed by a team that would like to adopt Panoptes by means of the provided reference implementation. The following subsections present the steps that a team's software engineers and data scientists need to take.

5.7.1 Software Engineers

The responsibilities for the software engineers are: setting up the base infrastructure, deploying the relevant Panoptes services and implementing and deploying new algorithm runtime and ac-

tion services. Regarding the infrastructure, the software engineers need to set up a Kubernetes cluster with KNative installed. To ease adoption, the authors' GitHub profile contains a repository[1] which contains the Kubernetes manifest necessary to install all the relevant KNative components on a Kubernetes cluster. Additionally, the manifests install all of the necessary Panoptes services as well as the example algorithm runtime and action services provided in the reference implementation. Onwards, the software engineers would be responsible for implementing new algorithm runtimes and actions that satisfy the requirements of their team. They should also document the algorithm runtimes to help data scientists implement their algorithms. For that, the Python runtime documentation[2] provided in the reference implementation can be used as a guide.

5.7.2 Data Scientists

The steps that data scientists need to follow are: becoming familiar with PDL, implementing dataset-shit-detection algorithms and finally using PDL to deploy their ML monitoring workflows. While the present thesis contains an in-depth documentation of PDL, the author's GitHub profile also contains a more succinct version[3] that might be more accessible for some. After becoming familiar with PDL, data scientists can start implementing their own algorithms to detect dataset shift. For that, they can leverage algorithm runtimes implemented by their software engineer colleagues or start with the example Python runtime provided in the reference implementation and documented in the provided GitHub repository. For convenience, the Python runtime also comes with a CLI tool[4] that allows data scientists to test algorithms under development on their local machines. The final step is to put it all together to start deploying ML monitoring workflows specified using PDL that periodically execute the developed algorithms to detect dataset shift and take mitigative action when needed.

[1] `https://github.com/pkourouklidis/panoptes-example-cluster`
[2] `https://github.com/pkourouklidis/panoptes-wiki/blob/main/Algorithm-Runtimes.md`
[3] `https://github.com/pkourouklidis/panoptes-wiki/`
[4] `https://github.com/pkourouklidis/python-function-runtime-cli`

5.8 Chapter Summary

This chapter presented the technical details of Panoptes' reference implementation to showcase how the architecture presented in chapter 3 can be implemented in practice. Additionally, the chapter presented the required steps that a team needs to take in order to adopt Panoptes. The reference implementation presented in this chapter was utilised in the empirical studies presented in the next chapter that were conducted to evaluate the proposed approach.

Chapter 6

Evaluation

For the evaluation of the proposed solution presented in this thesis, three separate empirical studies were conducted. The studies were designed based on the ABC framework for software engineering research [75]. This framework provides a taxonomy of eight archetypes based on the level of obtrusiveness of the research and the generalisability of the results. For this evaluation, three of these archetypes have been used: the judgement study, laboratory experiment and experimental simulation archetypes. All three studies were performed towards the end of the project. As such, their purpose was to evaluate rather than shape the proposed solution.

The following sections present each study's goals, explain the ABC framework archetype they are based upon, and discuss the results.

6.1 Initial Approach Validation

The first empirical study was conducted following the finalisation of PDL to verify that the domain has been modelled correctly and evaluate the solution's potential value for data scientists.

Specifically, the following research questions were formulated before the study was conducted:

RQ1 Is the proposed DSL sufficiently expressive for the modelled domain?

RQ2 How do data scientists evaluate the solution's potential for lowering the technical barrier in the dataset shift management domain?

RQ3 Does the developed set of constraints provide value through error prevention?

The study designed to answer the above research questions follows the *judgement study* archetype. In the ABC framework, judgement studies involve seeking input and opinions from experts in the field to evaluate or validate certain practices, methods, or concepts. These studies rely on the expertise and judgement of selected individuals with relevant experience and knowledge. The experts are typically presented with specific questions or scenarios, and their responses are analysed to gain insights and make informed decisions. Judgement studies are conducted in neutral settings that play no role in the study.

More concretely, the study was conducted in semi-structured interviews with the following format. Each participant was given a detailed introduction to the proposed solution through a one-to-one session with the author. PDL's metamodel was explained in depth, and they were shown samples of the textual syntax in the developed web editor, which provided some context about how they would be expected to interact with the system. Subsequently, a free-form discussion commenced with organically occurring questions about the system, as well as feedback and impressions. At the end of the session, to distil the participants' feedback in a standardised manner, they were given a number of prepared questions about the proposed solution and asked to answer with a positive, neutral or negative sentiment explicitly. Since this empirical study involved external participants, the University of York research ethics process was followed. More information about this process is provided in Appendix B.

The rest of the section presents the participants' demographics, followed by the questions asked and the results of the study.

6.1.1 Participant Demographics

The study's participants are eleven professional data scientists working in the Applied Research department of British Telecom (BT), where the author was also employed at the time of the

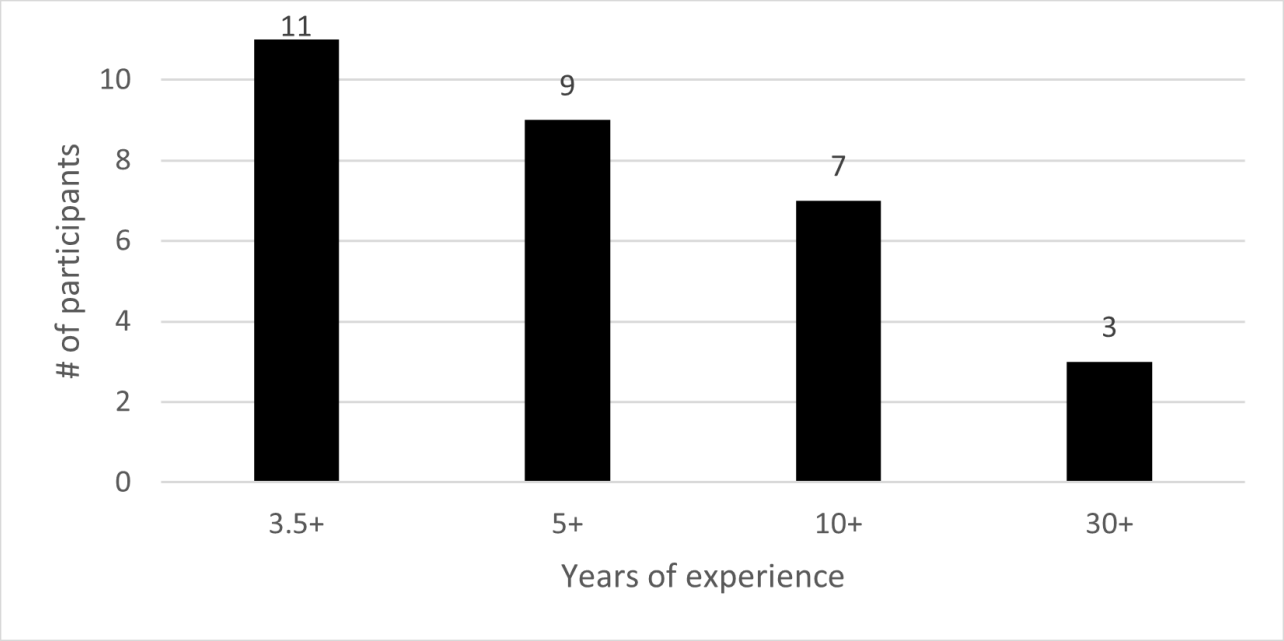


Figure 6.1: Participants' professional experience

study. As seen in Figure 6.1, all participants had a minimum of 3.5 years of professional experience in data science, with seven having ten or more years of experience and three having thirty or more years of experience.

Regarding academic qualifications, all participants held a degree in a quantitative field related to data science. Of the 11 participants, 2 held a Bachelor's degree, 3 held a Master's degree, and 6 held a Doctorate.

6.1.2 Study Questions and Results

In order to answer the first research question, the participants the following questions were asked:

RQ1.1 Could a model-based approach, in general, provide sufficient coverage of the dataset shift domain?

RQ1.2 Is PDL expressive enough to provide sufficient coverage of the dataset shift domain?

For both RQ1.1 and RQ1.2, 10 participants answered with a positive sentiment and 1 participant answered with a neutral sentiment.

In order to answer the second research question, the following questions were asked:

RQ2.1 Could a model-based workflow potentially lower the technical barrier for data scientists who would like to implement dataset-shift-related functionality?

RQ2.2 Does the proposed solution have the potential to lower the technical barrier for data scientists who would like to implement dataset-shift-related functionality?

For both RQ2.1 and RQ2.2, 10 participants answered with a positive sentiment and 1 participant answered with a neutral sentiment.

In order to answer research question number three, the following question was asked:

RQ3.1 Does model validation provide substantial value through error prevention?

For question RQ3.1, 10 participants answered with a positive sentiment and 1 participant answered with a neutral sentiment.

Finally, to document any reservations expressed by the participants and offer a response, the following question was asked: Are there any omissions in the metamodel or the functionality offered by the solution? The responses of the two participants who responded with a neutral sentiment to some of the questions offered the following insights.

The first participant expressed a neutral sentiment in the first two questions regarding domain coverage based on a particular use case they had in mind. More specifically, they posed a scenario in which, to determine the existence of dataset shift, one would need external information in addition to data handled by an ML platform (e.g. features, labels, predictions). This scenario could indeed be challenging to address whether one is using Panoptes or not. This kind of edge case could be addressed within the scope of Panoptes by adding a manual data retrieval process within an algorithm's implementation. This is suboptimal as it requires more effort to implement compared to a standard algorithm. However, it could be considered acceptable because it is only required in specific scenarios and would also be required if Panoptes was not

used. Ideally, all data needed to train ML models and detect dataset shift is handled by the ML Platform.

The second participant expressed a neutral sentiment in the last three questions regarding the lowering of the technical barrier when implementing dataset-shift-related functionality and the utility of model validation. It should be noted that this participant had an extensive software engineering background. It is therefore considered understandable that they did not perceive the potential of Panoptes for effort reduction as significant as the rest of the participants. The participant's main concern was related to the capacity of the solution to handle large amounts of data. The way that this is addressed within the scope of Panoptes is through the concept of algorithm runtimes. This allows for the usage of technologies that can handle large amounts of data while keeping the complexity exposed to data scientists at acceptable levels.

In conclusion, most of the participating data scientists expressed a positive sentiment and willingness to further experiment with PDL and the Panoptes system in general.

6.2 Hands-On Validation

This section presents the second empirical study conducted to evaluate the proposed solution, with the participation of 10 data scientists working within British Telecom (BT). Out of the 10 participant, 5 of them also participated in the first empirical study. For the rest of the participants, this was their first time interacting with Panoptes. The participants' years of experience and academic qualifications were similar to those in the first study, as they were recruited from the same pool of people. Explicit information about participant demographics was not collected for this study.

The study took place after the technical aspects of the solution were finalised, making it possible for the participants to have a first-hand experience with it. It sought to answer the following research questions:

RQ1 Are the concepts of the DSL easily understood by data scientists?

RQ2 Can data scientists effectively utilise the DSL to implement dataset shift detection strategies?

RQ3 How do data scientists evaluate the solution's potential for reducing the effort of implementing ML performance monitoring policies?

6.2.1 Study Design

The study was split into three parts, each seeking to answer one of the research questions.

The first two parts are in the form of a laboratory experiment in the ABC framework. Laboratory experiments are conducted in a controlled and contrived setting. These experiments aim to neutralise confounding factors and extraneous conditions, allowing researchers to exercise maximum precision in measuring behaviour on the studied object. Laboratory experiments often involve a limited number of subjects and discrete trials of relatively short time spans.

While the context in a laboratory experiment is removed from real-world software development environments, it provides a controlled environment to study human behaviour and performance in software-related tasks. Researchers can observe and measure how individuals or teams interact with software systems, tools, or processes and identify factors influencing their behaviour and performance.

First Part

More concretely, in the first part, the participants were given access to documentation material describing PDL, similar to Chapter 4. The documentation provided is publicly available[1]. After reading the material, the participants were shown the example PDL model in Listing 6.1, which utilises every feature of the DSL. After reading the PDL model, participants were given fifteen comprehension questions to answer to quantify how well data scientists can comprehend PDL concepts in a short amount of time.

[1]https://github.com/pkourouklidis/panoptes-wiki

The following are the comprehension questions that the participants were called to answer:

1. What is the name of the Model that the callcentre Deployment uses?
2. What are the names of the AlgorithmExecutions defined in the callcentre Deployment?
3. What is the name of the Algorithm used by the service-duration-shift AlgorithmExecution?
4. What is the URL of the git repo that contains the code for the Algorithm accuracy-check?
5. Which ActionExecution will be triggered in case dataset shift is detected by the service-duration-shift Algorithm Execution?
6. Which parameters does the retrain Action define?
7. In the retrainCallcentre ActionExecution, can we omit providing a value to the ioNames parameter? Please justify.
8. In the service-duration-shift AlgorithmExecution, is the value true appropriate for the pValue parameter? Please justify.
9. What is the statistical type of the live data input for the service-duration-shift AlgorithmExecution?
10. Could we have used the Feature is_solved as input to the service-duration-shift AlgorithmExecution without causing any warnings/errors? Please justify.
11. In a scenario where one day has passed since trigger t1 has been activated, but the Deployment has only served 900 requests, will AlgorithmExecution service-duration-shift run? Please justify.
12. In a scenario where no AlgorithmExecution has run yet, and callcentre-accuracy runs for the first time, will AlgorithmExecution ema-accuracy run? Please justify.
13. What are the inputs of the callcentre Deployment?

14. Based on the inputs of the callcentre Deployment, which features are allowed as input for the Model that callcentre uses?

15. How would we need to modify the inputs of the callcentre Deployment if we wanted to substitute the current Model that is being used with Model callcentre-alternative without causing any warnings/errors?

The above questions were selected to check the participants' comprehension of PDL in the following ways:

- A number of the questions check whether participants understand the way PDL's textual syntax is used to instantiate the various classes of the metamodel. To answer the first question, for example, one needs to understand where in the example model the "call-centre" *Deployment* is defined and which ML *Model* it refers to. Questions 1-6, 9 and 13 belong in this category.

- A number of questions check whether the participants understand PDL's model validation mechanism described in Section 4.4. Questions 7-8, 10 and 14-15 belong in this category.

- Finally, questions 11 and 12 check whether the participants understand PDL's algorithm execution scheduling mechanism described in Section 4.3.

Second Part

For the second part, based on the call centre scenario, a system was developed that simulates customers' calls. The system includes the dashboard shown in figure 6.2 that can be used for initiating simulations with different parameters such as how frequently calls are being made, the number and the skill of the call centre's workers and how patient customers are. The combination of these parameters affects each call's wait duration, service duration, issue resolution and, ultimately, customer satisfaction.

```
FeatureStore{
    entities call{keys callID}
    request data additional_data
    features
        wait_duration:continuous{requires entities call},
        service_duration:continuous{requires entities call},
        is_solved:categorical{requires entities call},
        additional_feature{requires request data additional_data}
    labels is_happy:categorical
}

Model callcentre-tree{
    uses wait_duration, service_duration, is_solved
    outputs happiness_prediction
    predicts is_happy
}
Model callcentre-alternative{
    uses wait_duration, service_duration, is_solved, additional_feature
    outputs happiness_prediction_alt
    predicts is_happy
}

BaseAlgorithmRuntime pythonFunction
BaseAlgorithm kolmogorov-smirnov{
  codebase "https://github.com/pkourouklidis/kolmogorov-smirnov-algorithm"
  runtime pythonFunction
  severity levels 2
  accepts only continuous
  parameters pValue:Real
}
BaseAlgorithm accuracy-check{
  codebase "https://github.com/pkourouklidis/accuracy-algorithm"
  runtime pythonFunction
  severity levels 2
  parameters threshold:Real
}

HigherOrderAlgorithmRuntime higherOrderPythonFunction
HigherOrderAlgorithm exponential-moving-average{
    codebase "https://github.com/pkourouklidis/ema-algorithm"
    runtime higherOrderPythonFunction
    parameters alpha:Real, mandatory threshold:Real
    severity levels 2
}

Action email{
    parameters mandatory email:String
}
Action retrain{
    parameters mandatory ioNames:String, mandatory containerImage:String
}

Deployment callcentre{
    inputs callID
    model callcentre-tree

    BaseAlgorithmExecution service-duration-shift{
        algorithm kolmogorov-smirnov
        live data service_duration
        historical data service_duration
        actions 1->emailMe
        parameter values pValue = 0.05
    }
    BaseAlgorithmExecution callcentre-accuracy{
        algorithm accuracy-check
        live data is_happy, happiness_prediction
        parameter values threshold = 0.80
    }
    HigherOrderAlgorithmExecution ema-accuracy{
        algorithm exponential-moving-average
        observed execution callcentre-accuracy
        min observations 3
        max  observations 3
        parameter values alpha = 0.5, threshold = 0.8
        actions 1->emailMe
    }
    ActionExecution emailMe{
        action email
        parameter values email=panagiotis.kourouklidis@bt.com
    }
    ActionExecution retrainCallcentre{
        action retrain
        parameter values ioNames="wait_duration,service_duration,is_solved,is_happy",
            containerImage="registry.docker.nat.bt.com/panoptes/callcentre-model-training:latest"
    }
    Trigger t1{
        every 1000 samples or every one day
        execute service-duration-shift
    }
    Trigger t2{
        every 100 labels
        execute callcentre-accuracy
    }
}
```

Listing 6.1: PDL model provided to participants.

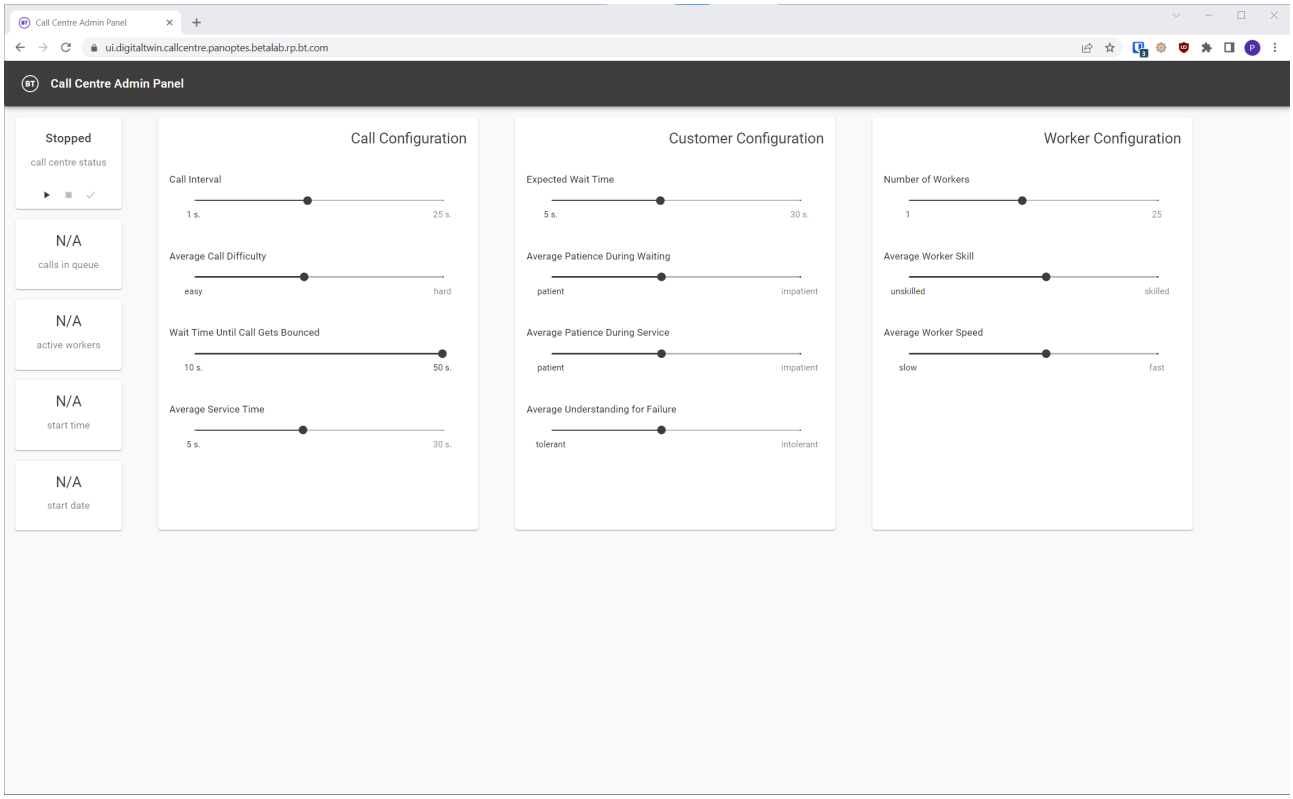


Figure 6.2: Simulation Dashboard

The simulator generates data for each call in the backend, including the ground truth of customer satisfaction. It also requests customer satisfaction predictions from an ML model trained for this task and deployed on the ML platform described in Chapter 5. This data, in addition to data from the orchestrator regarding algorithm execution results, is visualised on a separate dashboard developed for this purpose and shown in Figure 6.3.

Participants were given access to the dashboards and asked to complete the following tasks:

- To let participants familiarise themselves with the mechanics of the simulator, they were instructed to use the example PDL model and a set of settings for the simulator such that no dataset shift is observed. After executing the simulation, the users could observe in the visualisation dashboard that the algorithm execution detected no dataset shift.
- Keeping the PDL model unchanged, the participants were given a modified set of simulator settings that introduced covariate shift in the service duration feature. The algorithm execution defined in the given PDL model detected the shift, and participants received an email notification.

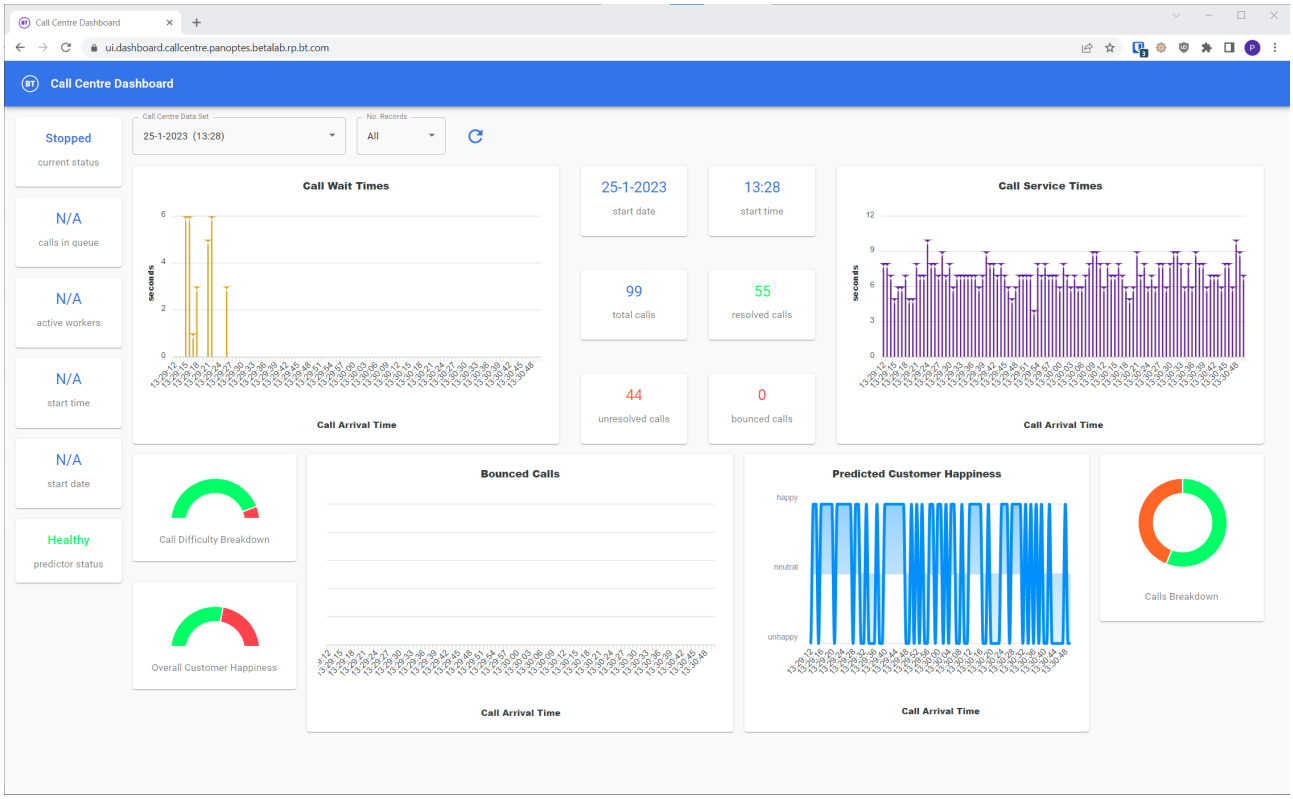


Figure 6.3: Overview Dashboard

- Finally, participants were given a set of simulator settings that changed the customers' patience to introduce concept shift. They were asked to modify the PDL model so that the shift would be detected and they would receive an email notification. An algorithm that detects concept shift based on the ML model's reduced accuracy on recent data was provided to the participants as the algorithm implementation portion was not the object of this study.

The PDL models created in the last step were collected and analysed to quantify how well data scientists utilised PDL for the given task. Listing 6.2 shows the correct answer to the given task.

One point was awarded for successfully completing each of the following subtasks:

1. Defining a *BaseAlgorithmExecution* that uses the "accuracy-check" *Algorithm*, included in the PDL model provided as a baseline. This can be seen in lines 40-43 of Listing 6.2.

2. Adding the necessary data as input to the *BaseAlgorithmExecution*. This can be seen in

```
FeatureStore{
  features
      wait_duration:continuous,
      service_duration:continuous,
      is_solved:categorical
  labels
      is_happy:categorical
}

Model callcentre-tree{
    uses wait_duration, service_duration, is_solved
    outputs happiness_prediction
    predicts is_happy
}

BaseAlgorithmRuntime pythonFunction

BaseAlgorithm kolmogorov-smirnov{
  codebase "https://github.com/pkourouklidis/kolmogorov-smirnov-algorithm"
  runtime pythonFunction
  severity levels 2
  accepts only continuous
  parameters pValue:Real
}

BaseAlgorithm accuracy-check{
  codebase "https://github.com/pkourouklidis/accuracy-algorithm"
  runtime pythonFunction
  severity levels 2
  parameters threshold:Real
}

Action email{
  parameters mandatory email:String
}

Deployment callcentre{
  model callcentre-tree

  BaseAlgorithmExecution concept_shift{
    algorithm accuracy-check
    live data happiness_prediction, is_happy
    actions 1->emailMe
  }

  ActionExecution emailMe{
    action email
    parameter values email=panagiotis.kourouklidis@bt.com
  }

  Trigger t1{
  every  100 samples
  execute concept_shift
  }
}
```

Listing 6.2: Correct Answer.

line 42 of Listing 6.2.

3. Adding the correct result to action execution map to the *BaseAlgorithmExecution*. This can be seen in line 43 of Listing 6.2.

4. Defining a *Trigger* with the newly created *BaseAlgorithmExecution* as the execution target. This can be seen in lines 51-54 of Listing 6.2.

5. Setting the frequency of the *Trigger* every 100 *Labels*. This can be seen in line 52 of Listing 6.2.

Third Part

The third part took the form of another judgment study. This qualitative form of evaluation was selected due to the difficulty of quantifying the effort required to build a monitoring solution and comparing it against a Panoptes-based solution with the same characteristics. Instead, five tasks were selected that were deemed representative of the monitoring domain. Participants were asked for their subjective evaluation in the following way: On a qualitative basis, how much do you think Panoptes could reduce the effort required for the following tasks? Please answer on a scale of one to five:

1. Implementing dataset shift algorithms.

2. Fetching the relevant data to check for dataset shift.

3. Scheduling the execution of dataset-shift-detecting algorithms.

4. Executing corrective actions in case of dataset shift.

5. Modifying various monitoring parameters (e.g. algorithm used, frequency of execution) on a live system.

Notably, the questions also address parts of the domain where we did not expect Panoptes to offer significant utility but were included to get a complete picture of the participants'

opinions. The implementation of dataset shift algorithms, for example, is streamlined by the runtime mechanism but still requires effort by data scientists.

6.2.2 Study results

Given that none of the participants had used PDL before, the results[2] indicate that data scientists can familiarise themselves with PDL relatively easily. They also expect Panoptes to provide significant benefits in terms of effort reduction for ML performance monitoring tasks.

Regarding the first two research questions, the participants spent approximately one hour reading the documentation material before answering the first questionnaire and completing the requested PDL model. Considering that all participants were busy professionals who could not afford to invest much time to complete the tasks, as participation was not part of their job responsibilities, the fact that most of the questions were answered correctly by every participant indicates that data scientists can pick up PDL very quickly.

Figure 6.4 shows the aggregate results for the questionnaire provided in the first part. In essence, it shows for each question how many participants answered it correctly. In addition, Figure 6.5 shows the number of mistakes for each individual participant. This Figure shows that except for one outlier, the errors are distributed among various participants and most participants made either zero or one mistake.

Figure 6.6 shows the aggregate results for the tasks given to the participants in the second part of the study. In addition, Figure 6.7 shows the number of mistakes for each individual participant. This shows that most participants were able to complete every task of the second part, with only two participants making one mistake and one participants making two.

Regarding the third research question, Figure 6.8 shows the cumulative score for every aspect participants were asked to evaluate. Since the evaluations were given on a scale of one to five, the maximum score was fifty. As expected, certain features of Panoptes were valued more than others. Specifically, the aspects requiring some degree of manual effort, namely developing new

[2]https://zenodo.org/record/8140392

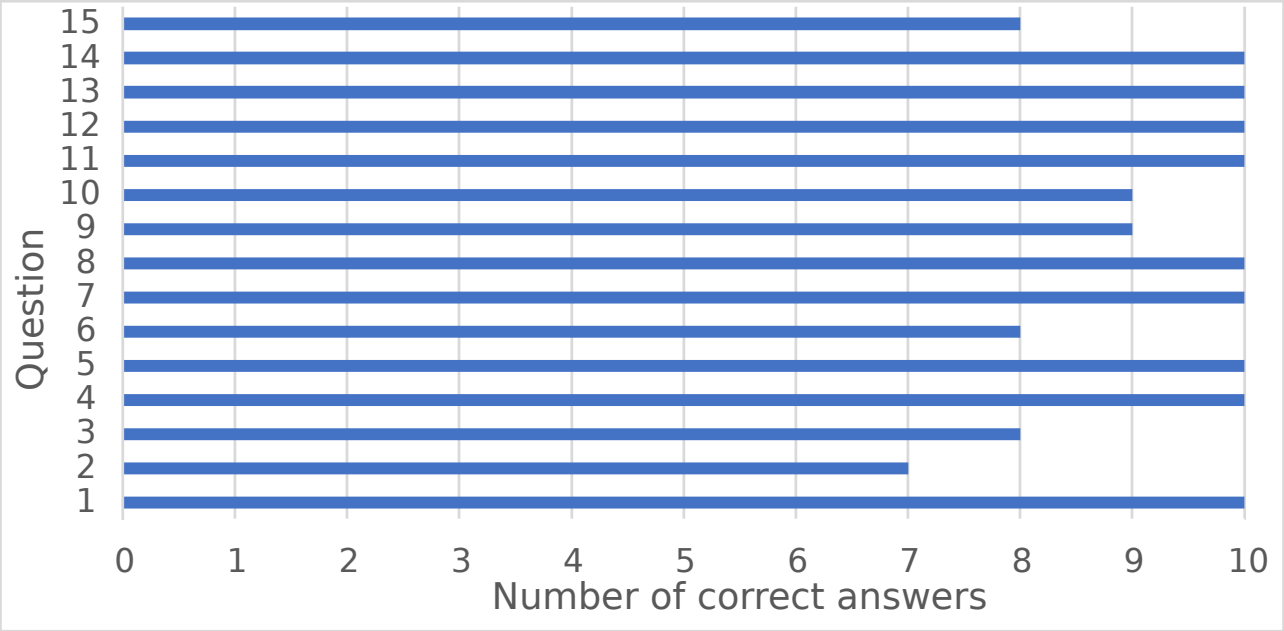


Figure 6.4: Aggregate results of the first questionnaire.

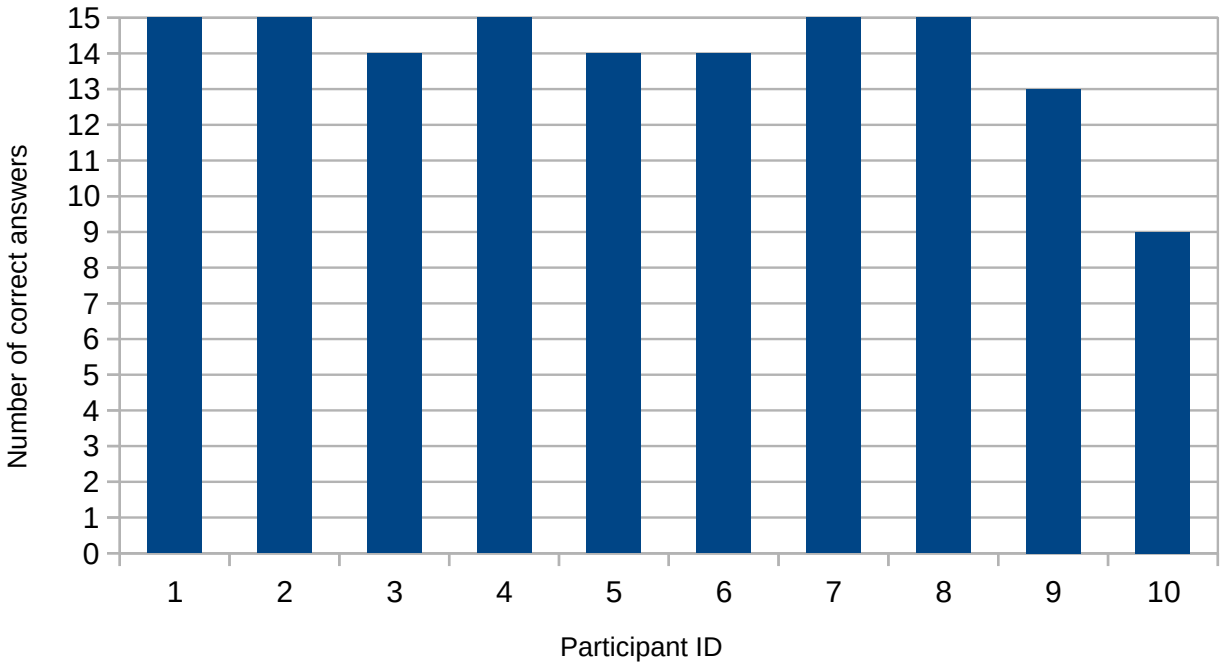


Figure 6.5: Individual participant results of the first questionnaire.

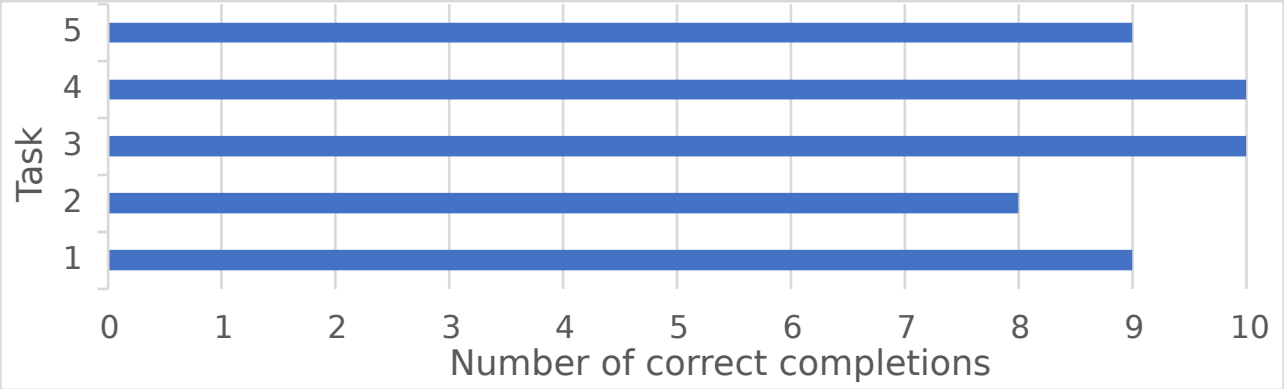


Figure 6.6: Aggregate results of PDL usage test.

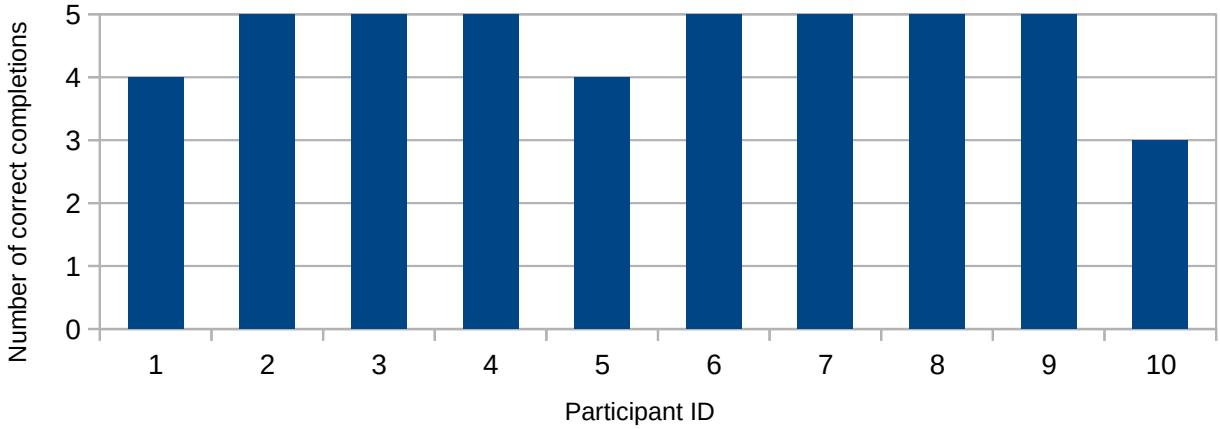


Figure 6.7: Individual participant results of PDL usage test.

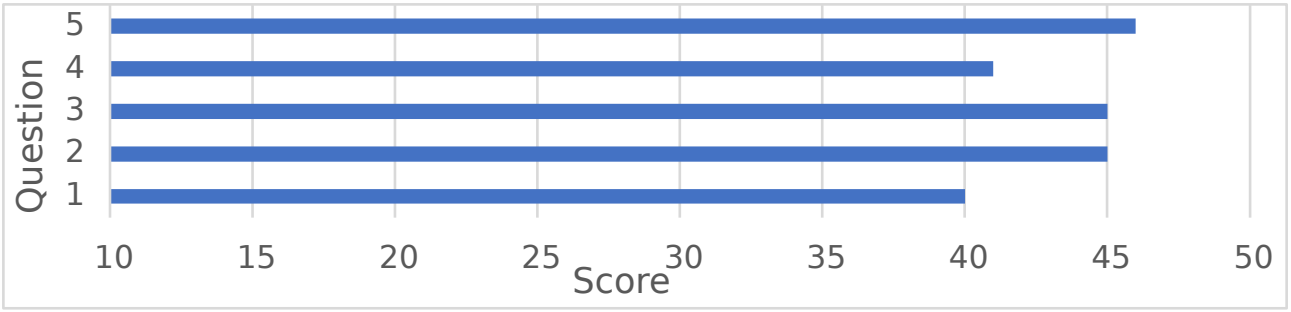


Figure 6.8: Aggregate effort reduction evaluation results.

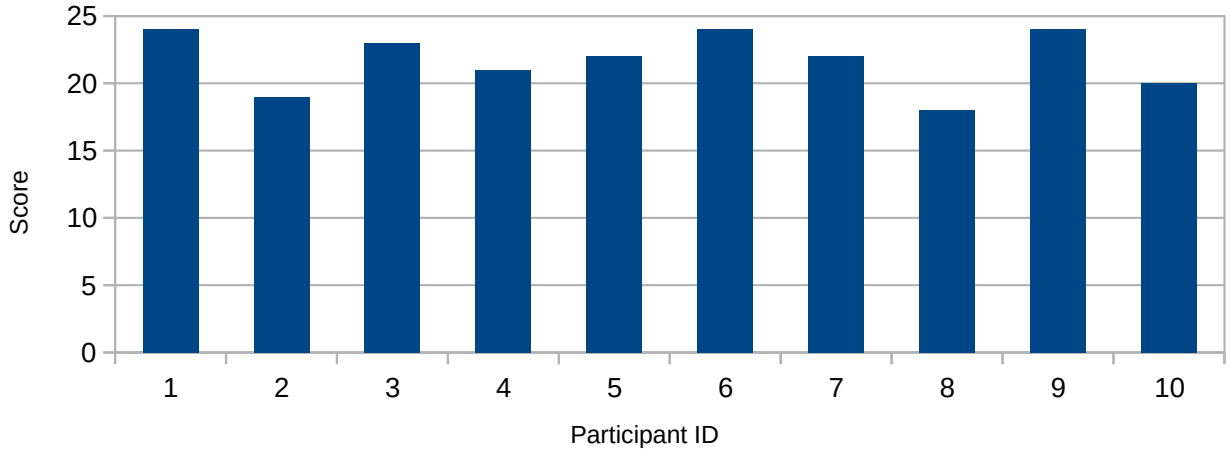


Figure 6.9: Individual participant effort reduction evaluation results.

algorithms and actions, received the lowest score. On the other hand, fully automated features, such as scheduling, received a higher score. In addition, Figure 6.9 shows the total score each individual participant gave for the five questions asked, for a maximum possible score of twenty-five. Once again, the results are closely distributed, with no participant evaluating Panoptes significantly higher or lower than the rest.

6.3 External ML Model Evaluation

The third empirical study examined the claim that Panoptes is general enough to support ML models of different input and output modalities and implementation technologies. For this purpose, three experimental simulations were conducted.

The ABC framework introduces experimental simulations as research strategies that aim to

observe the behaviour of a system in a controlled environment. They are suitable for situations where it is not feasible or practical to directly observe the system in question in a natural setting. They offer a way to study and evaluate system features without costly or time-consuming real-world implementations. Experimental simulations can provide insights into the behaviour and performance of the observed object.

More concretely, the three Panoptes-based monitoring systems were implemented for three ML model deployments of publicly available models developed by third parties. All three implementations are publicly available[3].

6.3.1 Image Classification

For the first experimental simulation, the EfficientNet-B0 image classifier was used [76]. As this model architecture has multiple publicly available implementations, the simulation was replicated twice based on a TensorFlow[77] and a Pytorch[78] implementation.

The model is trained on the ImageNet [79] dataset, but the Standford dogs [80] subset was used for this simulation due to its more manageable size. After deployment, an image processing bug that introduces covariate shift in the model's incoming image classification requests is simulated. The pixel values of one thousand randomly selected images were arbitrarily reduced in half to make them darker than what is typically seen in the dataset, thus introducing the shift.

To detect the introduced shift, a detection algorithm was developed for the "pythonFunction" algorithm runtime. It calculates the luminance of recent images and checks to see if they are distributed similarly to the luminances in the training set based on the Kolmogorov-Smirnov two-sample test. A second algorithm was also developed that utilises ground truth labels, when available, to calculate the model's accuracy for recent inputs and checks to see if it has dropped significantly below the model's expected accuracy. As EfficientNet-B0 achieves a 76.3% accuracy in ImageNet, an accuracy below 60% was chosen as indicative of a potential problem.

[3]https://github.com/pkourouklidis/panoptes

Listing 6.3 shows the PDL model used in this experiment.

6.3.2 Speech Recognition

For the second experimental simulation, the publically available Silero speech-to-text model was used [81]. The model's dataset is not provided, so as a proxy, the common voice dataset [82], a publically available voice dataset, was used. Similarly to the first experiment, we simulate a signal processing bug introducing dataset shift. To achieve this, 1000 randomly selected voice clips were captioned after white noise was added to them. To detect this shift, a detection algorithm that compared the signal-to-noise ratio of the distorted audio clips to those in the training set was developed. Additionally, a second algorithm that utilises ground truth data was also implemented. It calculates the word error rate, an accuracy metric commonly used for speech-to-text models, and checks to see if it is below what is typically expected from the model.

Listing 6.4 shows the PDL model used in this experiment.

6.3.3 Credit Scoring

For the third case study, a credit-scoring model available on Kaggle and implemented using scikit-learn was used [83]. The model was trained on a tabular dataset with multiple columns containing information such as an applicant's income bracket, savings, and employment status. To simulate dataset shift, one thousand samples were classified after the value of the "gender" column was switched for some of them. To detect this shift, a detection algorithm that calculates the L-infinity distance was implemented. Additionally, the accuracy calculation algorithm developed for the first case study was reused since it applies to both models.

Listing 6.5 shows the PDL model used in this experiment.

```
FeatureStore{
    features dog_input
    labels dog_label}

Model dogs-efficientnet{
    uses dog_input
    outputs dog_prediction
    predicts dog_label}

BaseAlgorithm luma-check{
    codebase "https://github.com/pkourouklidis/luma-check-algorithm"
    runtime pythonFunction
    severity levels 2
    parameters pValue:Real}

BaseAlgorithm accuracy-check{
  codebase "https://github.com/pkourouklidis/accuracy-algorithm"
  runtime pythonFunction
  severity levels 2
  parameters threshold:Real}

Deployment dogs{
    model dogs-efficientnet

    BaseAlgorithmExecution luma-check-dogs{
    algorithm luma-check
    live data dog_input
    historical data dog_input
    actions 1->emailMe
    parameter values pValue = 0.05
  }

  BaseAlgorithmExecution accuracy-check-dogs{
    algorithm accuracy-check
    live data dog_prediction, dog_label
    parameter values threshold = 0.6
    actions 1->emailMeDogs
  }

  ActionExecution emailMeDogs{
    action email
    parameter values email=panagiotis.kourouklidis@bt.com
  }

  Trigger s5{
  every 1000 samples
  execute luma-check-dogs
  }

  Trigger s6{
  every 1000 labels
  execute accuracy-check-dogs
  }
 }
```

Listing 6.3: PDL Model for the Image Classifier Experiment.

```
FeatureStore{
    features
        input
    labels
      label
}

Model stt-silero{
    uses input
    outputs prediction
    predicts label
}

BaseAlgorithm snr-check{
    codebase "https://github.com/pkourouklidis/SNR-check"
    runtime pythonFunction
    severity levels 2
    parameters pValue:Real
}

BaseAlgorithm wer-check{
    codebase "https://github.com/pkourouklidis/word-error-rate-algorithm"
    runtime pythonFunction
    severity levels 2
    parameters threshold:Real
}

Deployment stt{
    model stt-silero

    BaseAlgorithmExecution snr-check-stt{
    algorithm snr-check
    live data input
    historical data input
    actions 1->emailMe
    parameter values pValue = 0.05
  }

  BaseAlgorithmExecution wer-check-stt{
    algorithm wer-check
    live data prediction, label
    parameter values threshold = 0.3
    actions 1->emailMeStt
  }

  ActionExecution emailMeStt{
    action email
    parameter values email=panagiotis.kourouklidis@bt.com
  }

  Trigger t3{
  every
  1000 samples
  execute snr-check-stt
  }

  Trigger t4{
  every
  1000 labels
  execute wer-check-stt
  }

}
```

Listing 6.4: PDL Model for the Speech Recognition Experiment.

```
FeatureStore{
  features
      sex
  labels
      credit_label
}
Model credit-nb{
    uses sex
    outputs credit_prediction
    predicts credit_label
}
BaseAlgorithm l-infinity{
    codebase "https://github.com/pkourouklidis/l-infinity-algorithm"
    runtime pythonFunction
    severity levels 2
    parameters threshold:Real
}

BaseAlgorithm accuracy-check{
  codebase "https://github.com/pkourouklidis/accuracy-algorithm"
  runtime pythonFunction
  severity levels 2
  parameters threshold:Real
}

Deployment credit{
  model credit-nb

  BaseAlgorithmExecution l_inf_sex{
    algorithm l-infinity
    live data sex
    historical data sex
    actions 1->emailMe
    parameter values threshold = 0.10
  }

  BaseAlgorithmExecution credit-accuracy-check{
      algorithm accuracy-check
      live data credit_label, credit_prediction
      actions 1->emailMe
      parameter values threshold = 0.60
  }

  ActionExecution emailMe{
    action email
    parameter values email=panagiotis.kourouklidis@bt.com
  }


  Trigger t4{
  every
  1000 labels
  execute credit-accuracy-check
  }
}
```

Listing 6.5: PDL Model for the Credit Scoring Experiment.

6.4 Threats to Validity

While the empirical studies presented in this chapter provide a positive outlook on the validity of the proposed approach, it is important to take into consideration the various potential threats to their validity. In the following subsections these potential threats are presented in four distinct categories as defined by Cook and Campbell [84] and also presented in the context of software engineering by Wohlin et al. [85]

6.4.1 Conclusion Validity

As the name suggests, conclusion validity is concerned with whether a conclusion can be safely made on the basis of the experimental results. The main threat to the conclusion validity of the first two studies is the relatively low number of participants which could potentially limit the generalisability of the results. Similarly, in the third study, only a small number of ML models where utilised to evaluate the proposed approach.

6.4.2 Internal Validity

Threats to internal validity refers to factors affect the observed outcome but are unrelated to the treatment under study. In our first two studies, for example, threats to internal validity would be factors that affect participants' evaluation of Panoptes but are unrelated to the solution itself. On such factor is that study participants were all unpaid volunteers. As such, there is the possibility of a selection effect. In particular, out of all the individuals that were contacted to take part in the studies, the ones that accepted the invitation might be individual that are more receptive to novel solutions that target the ML monitoring domain compared with the average data scientist.

6.4.3 Construct Validity

Construct validity refers to whether the measured quantity of an experiment accurately reflects the construct under study. A construct validity threat in our case is the fact that the effort reduction for deploying ML monitoring workflow using Panoptes is only measured indirectly. Ideally, a group of data scientists would have been recruited and tasked with implementing a production-quality ML monitoring system with and without using Panoptes. Then, the number of engineering hours required in each case would be compared. Because this was considered infeasible, it was replaced with a questionnaire that asked data scientists to evaluate the solution's potential for effort reduction.

6.4.4 External Validity

External validity refers to whether results of an experiment generalise to non-experimental settings. More concretely, in the case of the the first two studies, all participating data scientists worked for BT. This might negatively affect the results' ability to generalise to the wider population of data scientists. In addition, while an effort was made to incorporate industrial best-practices when developing the simulation environment for the second experiment and the case studies in the third experiments, they are not actual production systems. There is therefore a risk that the results will not generalise to production settings.

6.5 Chapter Summary

In this chapter, the evaluation of the proposed solution, Panoptes, is presented. Three empirical studies were conducted to assess Panoptes' usability, domain coverage and potential to lower technical barriers for data scientists.

Overall, the results of the studies provided positive insights into the usability and effectiveness of Panoptes in the context of ML performance monitoring and dataset shift detection. The

results suggest that Panoptes has the potential to streamline monitoring tasks and lower the technical barriers for data scientists that deploy ML models in production environments.

Chapter 7

Conclusion

This thesis has investigated whether an MDE approach can be successfully applied in the novel domain of ML monitoring to address the challenges indicated by the relevant literature. The rest of the chapter is structured as follows: Section 7.1 enumerates the main contributions of the thesis and how they map to the objectives defined in Chapter 1. Section 7.2 discusses the perceived limitations of the proposed solution. Finally, Section 7.3 provides direction for future work.

7.1 Thesis Contributions

This thesis makes contributions to the state of practice across various axes. In terms of the developed solution, the following contributions are made:

- The design of PDL, a DSL narrowly focused on the ML monitoring domain. By narrowly focusing on one domain, the DSL can better cater to the domain experts, as shown by the positive usability evaluations. A detailed description of PDL has been published at SAM 2023 [3].
- Designing Panoptes, a system that combines the DSL with an architecture that delivers multiple desirable attributes. Firstly, the loose coupling of the MDE layer with the

underlying platform offers portability, a desirable attribute from a commercial advantage point of view. This attribute satisfies Objective 1c. Additionally, decoupling the algorithmic from the technical layers through a FaaS approach enables a separation of concerns between data scientists and software engineers. This separation lowers the technical barriers for data scientists and empowers them to deploy ML monitoring workflows autonomously. This attribute satisfies Objective 1a Finally, the architecture enables extensibility by making adding new algorithms and runtimes straightforward. This attribute satisfies Objective 1b. A detailed description of Panoptes has been published at SAM 2023 [3] and an earlier version of the solution has been published at MDE Intelligence 2021 [2].

- Providing a publicly available reference implementation for the proposed solution using popular open-source technologies further demonstrates the approach's feasibility and enables experimentation by others.

To support the validity of the proposed solution and achieve the remaining objectives of the thesis, three empirical studies were conducted to evaluate the solution's usability, domain coverage and technical barrier-lowering potential.

- Evaluating usability was part of Objective 2a. This evaluation took place during the second empirical study as presented in Section 6.2.

- Evaluating domain coverage was part of Objective 2b. This evaluation took place in the third empirical study as presented in Section 6.3. Domain coverage was also partly evaluated in the first empirical study since it was the study's first research question as explained in Section 6.1.

- Evaluating the solution's ability to lower technical barriers was part of Objective 2c. This was evaluated via the questionnaires given to data scientists in both the first and second empirical studies.

Additionally, from a non-technical point of view, the application of MDE for a novel domain within the wider AI governance domain uncovers new opportunities. This application raises awareness about the capabilities of MDE amongst a new audience who might not have been previously aware of it. For example, as mentioned in Section 3.1, people within BT's applied research department who have been collaborating with the TM Forum to develop a model for the AI governance domain in the form of a REST API were not aware of the MDE field prior to interacting with the author.

7.2 Solution Limitations

The following is a discussion about the perceived limitations of the proposed solution and potential avenues to overcome them in the future:

Adoption hurdles: While the conducted empirical studies indicate that data scientists become accustomed to the proposed solution quickly and even express positive opinions about it when probed, there still remains to be seen whether the solution's organic adoption will increase over time. It is possible that the solution's textual syntax could make data scientists perceive it as a burden due to it being yet another tool that one has to put effort into learning. The first future work avenue mentioned in the next Section explores a potential mitigation for this limitation.

Organisational Buy-in: Apart from data scientists, another factor that could hinder the adoption of the proposed solution is a potential reluctance from an organisations software engineers or higher-level executives. Even though an effort was made when designing the solution to hide the MDE-specific technologies for most software engineers, an organisation would still need some software engineers with MDE experience to maintain the MDE layer or to add new features to it. Therefore, organisations might be reluctant to adopt a new technology, given that the solution is narrowly focused on one domain. One potential avenue

to address this limitation would be to expand the domain which the solution addresses, as explained further in the following Section.

7.3 Future Work

7.3.1 Enabling Adoption

The current solution strives to make adoption easy for data scientists. Contributing to this is implementing a web-based editor, eliminating the need for a working Eclipse installation. Additionally, by making it easy for software engineers to add new runtimes, data scientists can implement their algorithms in the languages they prefer (e.g. Python or R) rather than the language that the solution happens to be implemented in (i.e. Java).

Building upon this idea, it would be possible to make the adoption of PDL even more accessible for data scientists by following an embedded DSL approach. Embedded DSLs, as explained in Subsection 2.4.2, are hosted within a general-purpose programming language and use its concrete syntax. This approach is enabled by the fact that PDL abstract and concrete syntaxes are decoupled, and, as explained in Section 4.5, the architecture allows even the usage of multiple concrete syntaxes simultaneously. Wasowski and Berger [86] provide an overview of the different strategies that can be leveraged to implement an embedded DSL.

7.3.2 Domain Expansion

Panoptes' architecture could be utilised to incrementally cover more of the AI governance domain. The distributed architecture is well-suited to incrementally onboarding new domains and slowly building credibility within an organisation as the technology is proven across multiple domains. The following are examples of domains that could potentially benefit from a Panoptes-style solution and are therefore suitable for future exploration.

ML Model Rollouts: A straightforward way to expand the domain that the proposed solution covers would be to include features that help data scientists roll out new ML models to replace the previous ones used in a deployment. Currently, the process by which replacement ML models are selected is not considered. One way, for example, that new ML models are selected is from a pool of candidate models via A/B testing or even multi-armed bandits testing. Further work would be needed to formalise this process as part of a DSL and designing the relevant technical architecture.

Regulatory Compliance: As more and more governments seek to regulate the usage of AI around the world, it is safe to assume that there is going to be a significant increase in the engineering resources needed to ensure that production AI systems are compliant. Relevant to this, Mökander [87] explores the field of AI auditing in the context of proposed regulations such as the EU's Artificial Intelligence Act. He draws parallels between AI auditing and financial auditing which has been legally mandated for decades. He also depicts the process of AI auditing as inherently multi-disciplinary. The above provides the motivation for the development of an MDE-based solution akin to Panoptes to streamline certain aspects of the auditing process. As a starting point, Bucaioni et al. [88] describe an MDE solution that automatically verifies the compliance of a concrete software system to a predefined reference implementation. This could potentially be extended to fit the AI auditing use case.

Appendix A

PDL Grammar

```
1 grammar org.lowcomote.panoptes.PanoptesX with org.eclipse.xtext.common.
    Terminals
2
3 import "http://www.lowcomote.org/panoptes/panoptesdsl"
4 import "http://www.eclipse.org/emf/2002/Ecore" as ecore
5
6 Platform returns Platform:
7   (
8     featureStore=FeatureStore |
9     mlModels+=Model |
10    deployments+=Deployment |
11    algorithms+=Algorithm |
12    algorithmRuntimes+=AlgorithmRuntime |
13    actions+=Action
14  )*
15 ;
16
17 Model returns Model:
18   {Model}
19   'Model'
20   name=(STRING|SAFESTRING)
21   '{'
22     (
```

```
23        ('uses' inputs+=[Feature|EString] ( "," inputs+=[Feature|EString])*) &
24        ('outputs' output=Prediction)
25      )
26    '}';
27
28 FeatureStore returns FeatureStore:
29    {FeatureStore}
30    'FeatureStore'
31    '{'
32      (
33        ('features' features+=Feature (',' features+=Feature)*)? &
34        ('entities' entities+=Entity ( "," entities+=Entity)*)? &
35        ('labels' labels+=Label (',' labels+=Label)*)? &
36        ('request' 'data' requestData+=RequestData (',' requestData+=
      RequestData)*)?
37      )
38    '}'
39 ;
40
41 Entity returns Entity:
42    name=(STRING|SAFESTRING)
43    '{'
44      'keys' keys+=Key ( "," keys+=Key)*
45    '}'
46    ;
47
48 ModelIO returns ModelIO:
49    Feature | Prediction | Label;
50
51 Feature returns Feature:
52    name=(STRING|SAFESTRING)
53    (':' type=statisticalVariableType)?
54    ('{'
55      (
```

```
56        ('requires' 'entities' entities+=[Entity|EString] ( "," entities+=[
      Entity|EString])*)? &
57        ('requires' 'request' 'data' requestData+=[RequestData|EString] ( ","
      requestData+=[RequestData|EString])*)?
58      )
59    '}')?
60 ;
61
62 Prediction returns Prediction:
63    {Prediction}
64    name=(STRING|SAFESTRING)
65    ('predicts' label=[Label|EString])?
66 ;
67
68 Label returns Label:
69    {Label}
70    name=(STRING|SAFESTRING)
71    (':' type=statisticalVariableType)?
72 ;
73
74 DeploymentIO returns DeploymentIO:
75    Key | RequestData;
76
77 RequestData returns RequestData:
78    {RequestData}
79    name=(STRING|SAFESTRING);
80
81 Key returns Key:
82    {Key}
83    name=(STRING|SAFESTRING);
84
85 enum statisticalVariableType returns statisticalVariableType:
86    continuous = 'continuous' | categorical = 'categorical' |
       orderedCategorical = 'ordered categorical';
87
```

```
88 Deployment returns Deployment:
89   'Deployment'
90   name=(STRING|SAFESTRING)
91   '{'
92     (
93       'model' mlModel=[Model|EString] |
94       ('inputs' inputs+=[DeploymentIO|EString] ( "," inputs+=[DeploymentIO|
     EString])* ) |
95       (algorithmexecutions+=AlgorithmExecution) |
96       (actionExecutions+=ActionExecution) |
97       (triggerGroups+=TriggerGroup)
98     )*
99
100  '}';
101
102 Algorithm returns Algorithm:
103   BaseAlgorithm | HigherOrderAlgorithm;
104
105 BaseAlgorithm returns BaseAlgorithm:
106   'BaseAlgorithm'
107   name=(STRING|SAFESTRING)
108   '{'
109     (
110       ('codebase' codebase=(STRING|SAFESTRING)) &
111       ('runtime' runtime=[BaseAlgorithmRuntime|EString]) &
112       ('severity' 'levels' driftLevels=EIntegerObject) &
113       ('accepts' (strict?='only')? supportedTypes+=statisticalVariableType (
      "," supportedTypes+=statisticalVariableType)*)? &
114       ('parameters' additionalParameters+=Parameter ( ","
     additionalParameters+=Parameter)*)?
115     )
116   '}';
117
118 HigherOrderAlgorithm returns HigherOrderAlgorithm:
119   {HigherOrderAlgorithm}
```

```
120   'HigherOrderAlgorithm'
121   name=(STRING|SAFESTRING)
122   '{'
123     (
124       ('codebase' codebase=(STRING|SAFESTRING)) &
125       ('runtime' runtime=[HigherOrderAlgorithmRuntime|EString]) &
126       ('severity' 'levels' driftLevels=EIntegerObject) &
127       ('parameters' additionalParameters+=Parameter ( ","
      additionalParameters+=Parameter)*)?
128     )
129   '}';
130
131 AlgorithmRuntime returns AlgorithmRuntime:
132   BaseAlgorithmRuntime|HigherOrderAlgorithmRuntime
133 ;
134
135 BaseAlgorithmRuntime returns BaseAlgorithmRuntime:
136   {BaseAlgorithmRuntime}
137   'BaseAlgorithmRuntime'
138   name=(STRING|SAFESTRING)
139   ('{'
140     'endpoint' endpoint=(STRING|SAFESTRING)
141   '}')?;
142
143 HigherOrderAlgorithmRuntime returns HigherOrderAlgorithmRuntime:
144   {HigherOrderAlgorithmRuntime}
145   'HigherOrderAlgorithmRuntime'
146   name=(STRING|SAFESTRING)
147   ('{'
148     'endpoint' endpoint=(STRING|SAFESTRING)
149   '}')?;
150
151 AlgorithmExecution returns AlgorithmExecution:
152   BaseAlgorithmExecution | HigherOrderAlgorithmExecution;
153
```

```
154 BaseAlgorithmExecution returns BaseAlgorithmExecution:
155   {BaseAlgorithmExecution}
156   'BaseAlgorithmExecution'
157   name=(STRING|SAFESTRING)
158   '{'
159     (
160       ('algorithm' algorithm=[BaseAlgorithm|EString]) &
161       ('live' 'data' currentIOValues+=[ModelIO|EString] ( ","
      currentIOValues+=[ModelIO|EString])*)? &
162       ('historical' 'data' historicIOValues+=[ModelIO|EString] ( ","
      historicIOValues+=[ModelIO|EString])*)? &
163       ('actions' ActionExecutionMap+=actionExecutionEntry ( ","
      ActionExecutionMap+=actionExecutionEntry)*)? &
164       ('parameter' 'values' parameterValueMap+=parameterValueEntry ( ","
      parameterValueMap+=parameterValueEntry)*)?
165     )
166
167   '}';
168
169 HigherOrderAlgorithmExecution returns HigherOrderAlgorithmExecution:
170   {HigherOrderAlgorithmExecution}
171   'HigherOrderAlgorithmExecution'
172   name=(STRING|SAFESTRING)
173   '{'
174     (
175       ('algorithm' algorithm=[HigherOrderAlgorithm|EString]) &
176       ('observed' 'execution' algorithmExecution=[AlgorithmExecution|EString
      ]) &
177       ('actions' ActionExecutionMap+=actionExecutionEntry ( ","
      ActionExecutionMap+=actionExecutionEntry)*)? &
178       (('minimum'|'min') 'observations' minDataPoints=EIntegerObject) &
179       (('maximum'|'max') 'observations' maxDataPoints=EIntegerObject) &
180       ('parameter' 'values' parameterValueMap+=parameterValueEntry ( ","
      parameterValueMap+=parameterValueEntry)*)?
181     )
```

```
182   '}';
183
184 actionExecutionEntry returns actionExecutionEntry:
185   {actionExecutionEntry}
186   key=EIntegerObject
187   '->'
188   value=[ActionExecution|EString]
189 ;
190
191 TriggerGroup returns TriggerGroup:
192   {TriggerGroup}
193   'Trigger'
194   name=(STRING|SAFESTRING)
195   '{'
196   compositeTriggers+=CompositeTrigger ("or" compositeTriggers+=
       CompositeTrigger)*
197   'execute' targets+=[BaseAlgorithmExecution|EString] ("," targets+=[
       BaseAlgorithmExecution|EString])*
198   '}'
199 ;
200
201 CompositeTrigger returns CompositeTrigger:
202   {CompositeTrigger}
203   'every'
204   (
205     (tt=TemporalTrigger)? &
206     (st=SampleBasedTrigger)? &
207     (pt=PredictionBasedTrigger)? &
208     (lt=LabelBasedTrigger)?
209   );
210
211 TemporalTrigger returns TemporalTrigger:
212   {TemporalTrigger}
213   ('one' frequency=frequencyEnum) | (cronString=STRING);
214
```

```
215 enum frequencyEnum returns frequencyEnum:
216   hourly = 'hour' | daily = 'day' | weekly = 'week' | monthly = 'month' |
       yearly = 'year';
217
218 SampleBasedTrigger returns SampleBasedTrigger:
219   {SampleBasedTrigger}
220   frequency=EIntegerObject 'samples';
221
222 PredictionBasedTrigger returns PredictionBasedTrigger:
223   {PredictionBasedTrigger}
224   frequency=EIntegerObject 'predictions';
225
226 LabelBasedTrigger returns LabelBasedTrigger:
227   {LabelBasedTrigger}
228   frequency=EIntegerObject 'labels';
229
230 Action returns Action:
231   {Action}
232   ('Action')
233   name=(STRING|SAFESTRING)
234   '{'
235     (
236       ('endpoint' endpoint=(STRING|SAFESTRING))? &
237       ('parameters' additionalParameters+=Parameter ( ','
       additionalParameters+=Parameter)*)?
238     )
239   '}';
240
241 Parameter returns Parameter:
242   {Parameter}
243   (mandatory?='mandatory')?
244   name=(STRING|SAFESTRING)
245    (':' type=parameterType)?
246 ;
247
```

```
248 enum parameterType returns parameterType:
249   Integer = 'Integer' | Real = 'Real' | String = 'String' | Boolean = '
       Boolean';
250
251 parameterValueEntry returns parameterValueEntry:
252   {parameterValueEntry}
253   key=(STRING|SAFESTRING)
254   '='
255   value=EString
256 ;
257
258 ActionExecution returns ActionExecution:
259   {ActionExecution}
260   ('ActionExecution')
261   name=(STRING|SAFESTRING)
262   '{'
263   (
264     ('action' action=[Action|EString]) &
265     ('parameter' 'values' parameterValueMap+=parameterValueEntry ( ","
       parameterValueMap+=parameterValueEntry)*)?
266   )
267   '}';
268
269 EIntegerObject returns ecore::EIntegerObject:
270   INT;
271
272 EString returns ecore::EString:
273   FLOAT | INT | STRING | SAFESTRING;
274
275 terminal FLOAT returns ecore::EFloat: '-'? ('0'..'9') '.' ('0'..'9')+;
276
277 @Override
278 terminal INT returns ecore::EInt: '-'? ('0'..'9')+;
279
```

```
280 terminal SAFESTRING: ('a'..'z'|'A'..'Z'|'_'|'.'|'@') ('a'..'z'|'A'..'Z'|'_
        '|'.'|'-'|'@'|'0'..'9')*;
```

Appendix B

Research Ethics Documents

Physical Sciences Ethics Committee Full Application Form

Section 1: PROJECT INFORMATION

1.1 Name and email of Principal Investigator (PI): Panagiotis Kourouklidis

1.2 Role of PI (Researcher / lecturer / Student): PhD student

1.3 Names and appointments of additional investigators: Professor Dimitris Kolovos (University of York), Dr. Joost Noppen (British Telecom), Dr. Nicholas Matragkas (CEA, France)

1.4 Title of the Project: MDE based system for ML performance monitoring

1.5 Funding source: Lowcomote, EU project

1.6 Where is the research taking place? British Telecom's applied research department. Ipswich, United Kingdom

1.7 Why are you applying for ethical approval (you must tick a box)?

- ■ It involves people
- ☐ It involves analysing unpublished data from or about living human beings?
- ☐ It involves animals?
- ☐ It involves data protection
- ☐ It is defence / military related
- ☐ There is a reputational risk to the University
- ☐ It restricts academic freedom
- ☐ It involves collaboration / partnership / funding from organisations tainted by ethically questionable activities?
- ☐ Other (please state) _____

1.8 Brief summary of the project and its key aims:
The project seeks to investigate novel techniques for monitoring the performance of ML models. We have created a domain specific language that can be used by data scientists to define monitoring strategies.

1.9 External Ethics Approval:
Has this work been approved by another external agency (collaborating institution, NHS) etc.

No

If yes please state where:

1.10 I have read and understood the University's Code of practice and principles for good ethical governance (https://www.york.ac.uk/staff/research/governance/research-policies/ethics-code/)

Yes

Section 2: HUMAN PARTICIPANTS

(If your project doesn't involve people please skip to section 3)

- *You must include an information sheet and consent form alongside this application form (or give a reason that they aren't needed).*
- *If a survey / questionnaire is being conducted please include this alongside the application.*
- *If you are conducting an online survey these can be embedded at the front of the survey and need to be included in this application.*

2.1 Who will your participants be? (Describe the criteria for inclusion / exclusion)

Data scientists employed British Telecom

2.2 Will they be paid? No

If yes how much (you must obtain a signed receipt of payment)?

2.3 Do any of the following apply?

Children (under 18)	No
If yes does the investigator have a current DBS check?	*N/A*
Vulnerable groups	No
The research is designed to be emotive or aversive	No
It involves taking bodily samples	No
Is physically invasive / challenging	No

If you answered yes to any of the above explain and justify the procedure and explain the steps taken to safeguard individuals:

2.4 Recruitment (How will you recruit participants?)

The PhD student is placed within BT as part of the Lowcomote project. He will reach out to the data scientist via email and ask them to participate if they so desire.

2.5 How will you guarantee anonymity? (This includes IP addresses and any identifying information)
If anonymity will not be provided explain why this is necessary.
Feedback about the developed system will be collected after the participants use it with the help of the researcher. The feedback collected will not be stored alongside each individual participant's personal information. After all the interviews are conducted, the evaluation data will be aggregated and included in a publication.

2.6 What types of personal data will you collect?
(https://www.york.ac.uk/records-management/dp/)

___ Physiological

___ Video footage / photographs

___ Audio (conversations, spoken tasks etc.)

___ Medical (in which case you are likely to need NHS approval)

X Personal (names, contact details etc.)

___ Financial

___ Other (Please state)___

How will you protect this data?
No personal data will be linked to a specific participant's feedback. Participants' contact details (e.g email addresses) were accessed through BT's directory on a machine managed by BT's IT department. Therefore, all contact details are protected by BT's organizational policies such as mandatory password protection on all laptops and full disk encryption. No data will ever migrate from a secure machine to an insecure personal device.

Section 3: DATA STORAGE AND TRANSMISSION

3.1 I have read and understood the University of York's Data Protection Policy (https://www.york.ac.uk/records-management/dp/policy/)

Yes

3.2 I will keep any data appropriately secure (e.g. in a locked cabinet), maintaining confidentiality and anonymity (e.g. identifiers will be encoded and the code available on a need to know basis) where possible.

Yes

3.3 Please describe the special precautions will you take to ensure anonymity when linking identifiable data to experimental data:

We will not be storing any personally identifying information.

3.4 If your data can be traced to identifiable participants/computer/address:

a) who will be able to access your data? N/A

b) approximately how long will you need to keep it in this identifiable format? N/A

3.5 If your project requires deviation from traditional data protection practices in research, or raises particular data protection issue please explain here: N/A

3.6 STUDENTS ONLY: Will any identifying data be kept securely by supervisors?

No

If No state why:

We will not be storing any personally identifying information.

Section 4: RISK ASSESSMENT

4.1 Has a departmental risk assessment been completed for this project, if appropriate?

No

If no why not?

No major risks have been identified

Section 5: ACADEMIC FREEDOM

5.1 Is there a secrecy clause to the research? No

If yes give details:

Section 6: REPUTATIONAL RISK *(if associated with a collaborative partner see section 7)*

6.1 Why is it appropriate for the University to be associated with this project? *(please also state what action has been taken to mitigate against potential reputational risk)*

This is a standard evaluation procedure of a software artifact with no reputational risk to the University.

Section 7: COLLABORATION WITH QUESTIONABLE ETHICAL STANDARDS / ACTIVITIES

7.1 Explain the nature of the collaboration / partnership and what about the organisation is ethically questionable:
To the best of our knowledge there is nothing ethically questionable about British telecom.

7.2 Why is it appropriate for the University to be associated with this organisation?
(please also state what action has been taken to mitigate against potential reputational risk)
The student's PhD project is being carried out in collaboration with British Telecom as part of the EU funded Lowcomote project.

Section 8: COMPLETION

The project team have read and understood this application:

Signed (PI): Panagiotis Kourouklidis Date: 20/04/2022

Additional Information
You may find the following codes of ethical practice and conduct relevant to your project:

British Psychological Society code of conduct:
http://www.bps.org.uk/the-society/code-of-conduct/code-ofconduct_home.cfm
IEEE
http://www.ieee.org/about/corporate/governance/p7-8.html
IET
http://www.theiet.org/membership/career/ethics/

Royal Academy of Engineering
http://www.raeng.org.uk/policy/engineering-ethics/ethics

The Royal Society
https://royalsociety.org/topics-policy/ethics-conduct/

Computer Science Department

Project title: MDE based system for ML performance monitoring
Participant Consent Form

Thank you for your interest in this project. The project seeks to investigate novel techniques for monitoring the performance of ML models. Specifically, we have created a domain specific language that can be used by data scientists to define monitoring strategies that we now seek to evaluate.

Please read the following statements carefully and tick the appropriate box (YES / NO):

I have read the information sheet about this project

I agree to take part in this project

I consent to being interviewed for this project

I understand my right to withdraw and/or destroy my data from

this project at any time

I am over the age of 18

I agree that data I provide (e.g., direct quotations) may be

published in anonymised form

Participant Name: _____

Participant Signature:_____

Date: _____/_____/_____

Researcher Name: _____

Researcher Signature: _____

Date: _____/_____/_____

If you wish to be informed about the outcomes from this project, please provide your email

address:_____

Physical Sciences Ethics Committee

Date: 17/05/2022
To: Panagiotis Kourouklidis
Subject: Ethics Approval
PSEC Application Ref: Kourouklidis20220424

Dear Panagiotis,

Thank you for your recent application to the Physical Sciences Ethics Committee for the project titled "MDE based system for ML performance monitoring". I am pleased to inform you that the committee has reviewed your application and supporting documents and we approve the research to be conducted on the basis described on the application form.

Please note the committee must be informed of any amendments to the protocol, participant information, or informed consent prior to the research taking place. Please follow the amendments procedures using the reference above in any correspondence concerning this project in the future.

It is also your responsibility to ensure continuing conformance to the University of York's ethical policy over the lifetime of the project.

Yours sincerely,

Dimitar Kazakov
PSEC Committee Member, Dept. Computer Science

On behalf of the Physical Sciences Ethics Committee

Bibliography

[1] P. Kourouklidis, D. Kolovos, N. Matragkas, and J. Noppen, "Towards a low-code solution for monitoring machine learning model performance," in *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pp. 1–8, 2020.

[2] P. Kourouklidis, D. Kolovos, J. Noppen, and N. Matragkas, "A model-driven engineering approach for monitoring machine learning models," in *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pp. 160–164, 2021.

[3] P. Kourouklidis, D. Kolovos, J. Noppen, and N. Matragkas, "A domain-specific language for monitoring ml model performance," in *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pp. 266–275, 2023.

[4] S. Amershi, A. Begel, C. Bird, R. DeLine, H. C. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: a case study," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019* (H. Sharp and M. Whalen, eds.), pp. 291–300, IEEE / ACM, 2019.

[5] M. D. Rossetti, *Simulation modeling and Arena*. John Wiley & Sons, 2015.

[6] N. Nahar, S. Zhou, G. Lewis, and C. Kästner, "Collaboration challenges in building ml-enabled systems: Communication, documentation, engineering, and process," in *Proceedings of the 44th International Conference on Software Engineering*, pp. 413–425, 2022.

[7] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. LIX, no. 236, pp. 433–460, 1950.

[8] N. J. Nilsson, *Artificial intelligence: a new synthesis*. Morgan Kaufmann, 1998.

[9] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[10] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[11] J. Markoff, *Machines of loving grace: The quest for common ground between humans and robots*. HarperCollins Publishers, 2016.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[13] N. J. Nilsson, *The quest for artificial intelligence*. Cambridge University Press, 2009.

[14] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, *et al.*, "Building watson: An overview of the deepqa project," *AI magazine*, vol. 31, no. 3, pp. 59–79, 2010.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[18] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT press, 2018.

[19] Gartner, "Types of machine learning report." `https://www.gartner.com/smarterwithgartner/understand-3-key-types-of-machine-learning`. Accessed: 2023-09-26.

[20] V. Vapnik, *The Nature of Statistical Learning Theory*. Statistics for Engineering and Information Science, Springer, 2000.

[21] J. Friedman, T. Hastie, R. Tibshirani, *et al.*, *The elements of statistical learning*, vol. 1. Springer series in statistics New York, 2001.

[22] J. C. Schlimmer and R. H. Granger, "Incremental learning from noisy data," *Mach. Learn.*, vol. 1, no. 3, pp. 317–354, 1986.

[23] B. Schölkopf, D. Janzing, J. Peters, E. Sgouritsa, K. Zhang, and J. Mooij, "On causal and anticausal learning," in *29th International Conference on Machine Learning (ICML 2012)*, pp. 1255–1262, International Machine Learning Society, 2012.

[24] M. Salganicoff, "Tolerating concept and sampling shift in lazy learning using prediction error context switching," *Artif. Intell. Rev.*, vol. 11, no. 1-5, pp. 133–155, 1997.

[25] A. Storkey, "When training and test sets are different: characterizing learning transfer," *Dataset shift in machine learning*, vol. 30, pp. 3–28, 2009.

[26] J. Quiñonero-Candela, M. Sugiyama, N. D. Lawrence, and A. Schwaighofer, *Dataset shift in machine learning*. MIT Press, 2009.

[27] M. Kull and P. Flach, "Patterns of dataset shift," in *First International Workshop on Learning over Multiple Contexts (LMCE) at ECML-PKDD*, 2014.

[28] J. G. Moreno-Torres, T. Raeder, R. Alaíz-Rodríguez, N. V. Chawla, and F. Herrera, "A unifying view on dataset shift in classification," *Pattern Recognit.*, vol. 45, no. 1, pp. 521–530, 2012.

[29] T. Fawcett and P. A. Flach, "A response to Webb and Ting's on the application of ROC analysis to predict classification performance under varying class distributions," *Machine Learning*, vol. 58, no. 1, pp. 33–38, 2005.

[30] E. Ameisen, *Building Machine Learning Powered Applications: Going from Idea to Product.* " O'Reilly Media, Inc.", 2020.

[31] Google, "Google's MLOps best practices." `https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning`. Accessed: 2023-09-26.

[32] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE transactions on knowledge and data engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.

[33] J. Zenisek, F. Holzinger, and M. Affenzeller, "Machine learning based concept drift detection for predictive maintenance," *Computers & Industrial Engineering*, vol. 137, p. 106031, 2019.

[34] S. Ackerman, O. Raz, M. Zalmanovici, and A. Zlotnick, "Automatically detecting data drift in machine learning classifiers," *arXiv preprint arXiv:2111.05672*, 2021.

[35] A. Soin, J. Merkow, J. Long, J. P. Cohen, S. Saligrama, S. Kaiser, S. Borg, I. Tarapov, and M. P. Lungren, "Chexstray: real-time multi-modal data concordance for drift detection in medical imaging ai," *arXiv preprint arXiv:2202.02833*, 2022.

[36] S. Ackerman, E. Farchi, O. Raz, M. Zalmanovici, and P. Dube, "Detection of data drift and outliers affecting machine learning model performance over time," 12 2020.

[37] S. Mirza, V. D. Nguyen, P. Mantini, and S. K. Shah, "Data quality aware approaches for addressing model drift of semantic segmentation models," *arXiv preprint arXiv:2402.07258*, 2024.

[38] Delloite, "State of AI 2022 report." `https://www2.deloitte.com/content/dam/Deloitte/us/Documents/deloitte-analytics/us-ai-institute-state-of-ai-fifth-edition.pdf`. Accessed: 2023-06-26.

[39] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Bießmann, and A. Grafberger, "Automating large-scale data quality verification," *Proc. VLDB Endow.*, vol. 11, no. 12, pp. 1781–1794, 2018.

[40] N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data lifecycle challenges in production machine learning: A survey," *SIGMOD Rec.*, vol. 47, no. 2, pp. 17–28, 2018.

[41] E. Breck, N. Polyzotis, S. Roy, S. E. Whang, and M. Zinkevich, "Data validation for machine learning," in *Conference on Systems and Machine Learning (SysML). https://www.sysml. cc/doc/2019/167. pdf*, 2019.

[42] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 2503–2511, 2015.

[43] M. Vartak, H. Subramanyam, W. Lee, S. Viswanathan, S. Husnoo, S. Madden, and M. Zaharia, "Modeldb: a system for machine learning model management," in *Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2016, San Francisco, CA, USA, June 26 - July 01, 2016* (C. Binnig, A. Fekete, and A. Nandi, eds.), p. 14, ACM, 2016.

[44] S. Schelter, J.-H. Boese, J. Kirschnick, T. Klein, and S. Seufert, "Automatically tracking metadata and provenance of machine learning experiments," in *Machine Learning Systems Workshop at NIPS*, pp. 27–29, 2017.

[45] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo, "Openml: networked science in machine learning," *SIGKDD Explorations*, vol. 15, no. 2, pp. 49–60, 2013.

[46] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke, "Tensorflow-serving: Flexible, high-performance ML serving," *CoRR*, vol. abs/1712.06139, 2017.

[47] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, *et al.*, "Accelerating the machine learning lifecycle with mlflow.," *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 39–45, 2018.

[48] S. Schelter, F. Bießmann, T. Januschowski, D. Salinas, S. Seufert, and G. Szarvas, "On challenges in machine learning model management," *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 5–15, 2018.

[49] K. M. Hazelwood, S. Bird, D. M. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang, "Applied machine learning at facebook: A datacenter infrastructure perspective," in *IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24-28, 2018*, pp. 620–629, IEEE Computer Society, 2018.

[50] Amazon, "Sagemaker documentation." `https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor.html`. Accessed: 2022-08-05.

[51] Microsoft, "Azure ml." `https://docs.microsoft.com/en-us/azure/machine-learning/`. Accessed: 2022-08-05.

[52] Google, "Vertex AI documentation." `https://cloud.google.com/vertex-ai/docs/model-monitoring`. Accessed: 2022-08-05.

[53] Google, "Vertex AI supported algorithms." https://cloud.google.com/vertex-ai/docs/model-monitoring/overview#calculating-skew-and-drift. Accessed: 2022-08-05.

[54] Microsoft, "Azure ml supported algorithms." https://docs.microsoft.com/en-us/azure/machine-learning/how-to-monitor-datasets#feature-details. Accessed: 2022-08-05.

[55] Gartner, "Future of ai technologies report." https://www.gartner.com/smarterwithgartner/gartner-predicts-the-future-of-ai-technologies. Accessed: 2023-06-26.

[56] B. Boehm and P. Papaccio, "Understanding and controlling software costs," *IEEE Transactions on Software Engineering*, vol. 14, no. 10, pp. 1462–1477, 1988.

[57] C. Gane, *Computer-aided software engineering: the methodologies, the products, the future.* Prentice-Hall, Inc., 1988.

[58] G. Premkumar and M. Potter, "Adoption of computer aided software engineering (case) technology: an innovation adoption perspective," *ACM SIGMIS Database: the DATABASE for Advances in Information Systems*, vol. 26, no. 2-3, pp. 105–124, 1995.

[59] D. N. Card, F. E. Mc Garry, and G. T. Page, "Evaluating software engineering technologies," *IEEE Transactions on Software Engineering*, no. 7, pp. 845–851, 1987.

[60] R. E. Yellen, "Systems analysts performance using case versus manual methods," in *Twenty-Third Annual Hawaii International Conference on System Sciences*, vol. 4, pp. 497–501, IEEE Computer Society, 1990.

[61] OMG, "Omg official website." https://www.omg.org/index.htm. Accessed: 2023-05-19.

[62] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synthesis lectures on software engineering*, vol. 3, no. 1, pp. 1–207, 2017.

[63] Epsilon developers, "Epsilon's industrial users." https://eclipse.dev/epsilon/users/. Accessed: 2023-09-26.

[64] M. Fowler, *Domain-specific languages*. Pearson Education, 2010.

[65] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, 2nd ed., 2009.

[66] S. Efftinge and M. Völter, "oAW xText: A framework for textual dsls," in *Workshop on Modeling Symposium at Eclipse Summit*, vol. 32, 2006.

[67] BT, "BT's 2023 annual report." `https://www.bt.com/bt-plc/assets/documents/investors/financial-reporting-and-news/annual-reports/2023/2023-bt-group-plc-annual-report.pdf`. Accessed: 2023-06-11.

[68] M. Mäntymäki, M. Minkkinen, T. Birkstedt, and M. Viljanen, "Defining organizational ai governance," *AI and Ethics*, vol. 2, no. 4, pp. 603–609, 2022.

[69] TM forum, "TM forum member list." `https://www.tmforum.org/membership/current-members/`. Accessed: 2022-08-05.

[70] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

[71] J. L. Hodges, "The significance probability of the smirnov two-sample test," *Arkiv för Matematik*, vol. 3, no. 5, pp. 469–486, 1958.

[72] T. B. Arnold and J. W. Emerson, "Nonparametric goodness-of-fit tests for discrete null distributions.," *R Journal*, vol. 3, no. 2, 2011.

[73] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[74] Pip developers, "Pip documentation." https://pip.pypa.io/en/stable/. Accessed: 2022-08-05.

[75] K. Stol and B. Fitzgerald, "The ABC of software engineering research," *ACM Trans. Softw. Eng. Methodol.*, vol. 27, no. 3, pp. 11:1–11:51, 2018.

[76] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, pp. 6105–6114, PMLR, 2019.

[77] TensorFlow team, "Efficientnet B0 implementation in TensorFlow." https://tfhub.dev/tensorflow/efficientnet/b0/classification/1. Accessed: 2023-09-26.

[78] Pytorch team, "Efficientnet B0 implementation in Pytorch." https://colab.research.google.com/drive/1Jw28xZ1NJq4Cja4jLe6tJ6_F5lCzElb4. Accessed: 2023-09-26.

[79] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[80] A. Khosla, N. Jayadevaprakash, B. Yao, and F.-F. Li, "Novel dataset for fine-grained image categorization: Stanford dogs," in *Proc. CVPR workshop on fine-grained visual categorization (FGVC)*, vol. 2, Citeseer, 2011.

[81] Silero Team, "Silero models: pre-trained enterprise-grade stt / tts models and benchmarks." https://github.com/snakers4/silero-models, 2021.

[82] Mozilla foundation, "Common voice dataset." https://commonvoice.mozilla.org/en. Accessed: 2023-09-26.

[83] Kaggle, "Credit scoring model." https://www.kaggle.com/code/kabure/predicting-credit-risk-model-pipeline. Accessed: 2023-09-26.

[84] T. Cook and D. Campbell, *Quasi-Experimentation: Design and Analysis Issues for Field Settings*. Houghton Mifflin, 1979.

[85] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[86] A. Wasowski and T. Berger, *Internal Domain-Specific Languages*. Springer International Publishing, 2023.

[87] J. Mökander, "Auditing of ai: Legal, ethical and technical approaches," *Digital Society*, vol. 2, p. 49, 2023.

[88] A. Bucaioni, A. Di Salle, L. Iovino, I. Malavolta, and P. Pelliccione, "Reference architectures modelling and compliance checking," *Software and Systems Modeling*, vol. 22, pp. 891–917, 2022.