CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Applying software engineering and machine learning practices to manage machine learning complexity

Master's thesis in Computer science and engineering

Sara Hillström

Johan Mejborn

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Applying software engineering and machine learning practices to manage machine learning complexity

SARA HILLSTRÖM
JOHAN MEJBORN

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Applying software engineering and
machine learning practices to manage machine learning complexity
Sara Hillström
Johan Mejborn

Supervisor: Richard Torkar, Department of Computer Science and Engineering
Advisor: Lovisa Bergström, Bonnier News
Examiner: Daniel Strüber, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2022

Applying software engineering and
machine learning practices to manage machine learning complexity
Sara Hillström
Johan Mejborn
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Today, both software engineering (SE) and machine learning (ML) are two fairly well-established areas within engineering. The field of software engineering for machine learning (SE4ML) addresses the issue of applying software engineering practices for software containing ML. Complexity is a term with a widespread definition, and the way of handling and defining it is something that differs between traditional software engineering and machine learning. In this thesis, complexity is defined as the measure of the resources expended by another system in interacting with a piece of software. If the interacting system is another machine we define it as resource cost, and if the interacting system is instead people (tasks such as, e.g., debugging and testing) we define it as software complexity.
This thesis was conducted in close collaboration with a partner company and aims to contribute to SE4ML by providing a framework aimed to act as guidance for how software complexity and resource cost may be addressed in different parts of the ML development process. The framework should also provide insights into possible trade-offs between software complexity and resource cost. To validate the framework, validation interviews with practitioners as well as representatives from academia were held, and the framework was also applied to an existing problem at the partner company. The latter was done by tweaking an existing ML model and developing two other models for comparison purposes.
In conclusion, the validation interviews and the application to an existing ML model confirmed that the framework is useful for practitioners. There are trade-offs between some of the different activities that form the framework, referred to as artifacts. This means that practitioners, to some extent, need to balance contradicting artifacts to optimize the resource cost and software complexity trade-off, depending on the specific use case at hand.

Keywords: software engineering, machine learning, complexity, framework.

Acknowledgements

We would like to thank our Chalmers supervisor, Richard Torkar, who has provided us with valuable input, challenged us whenever necessary, and given us access to his colleagues at the Software Engineering Division. We would also like to thank our supervisor at Bonnier News, Lovisa Bergström, and the ML team representatives Lukas Borggren and Emma Miléus, for their great support, for believing in us, and for giving us the freedom to explore our ideas.
Finally, we would like to thank the examiner Daniel Strüber for his feedback and for helping to improve the quality of the final report.

Sara Hillström & Johan Mejborn, Gothenburg, June 2022

Contents

List of Figures

List of Tables

1
Introduction

As more and more machine learning (ML) applications are deployed by various practitioners, there are plenty of best practices on how to successfully develop models, tune parameters, and increase performance. However, it is difficult to adopt software engineering (SE) best practices for ML applications compared to traditional software applications as they differ in fundamental ways [1, 2]. The field of software engineering for machine learning (SE4ML) addresses this issue of applying software engineering best practices for software containing ML [2]. Handling and defining complexity is one area that differs between traditional SE and ML.

Kearney et al. [3] use a definition for software complexity originally presented by Basili [4] already in 1980; *Complexity is the measure of the resources expended by another system in interacting with a piece of software.* If the interacting system is another machine the complexity is defined by the execution time and storage needed to conduct the computation [3], which is defined as *resource cost* in this paper. If the interacting system consists of people instead, then the complexity is defined by the difficulty to perform tasks such as debugging, testing, or coding [3] (i.e., *software complexity*).

To get practitioner input and to validate our work in a company context, this thesis was written in collaboration with Bonnier News. Bonnier News is one of the largest media houses in Sweden [5]. Every day they reach three million readers with the various brands in their portfolio [6]. Bonnier News offers a portfolio of digital products with the mission to provide people with truthful and high-quality news. Throughout the paper, Bonnier News will be referred to as the partner company.

1.1 Purpose and research questions

With this thesis, we aim to contribute to SE4ML, by researching SE and ML resource cost and software complexity as defined below, and based on this research provide a framework for ML engineers to use as guidance throughout the development process to tackle model complexity.

Definition 1.1.1 (Resource cost)**.** The measure of the resources expended by another system in interacting with a piece of software, where the interacting system is another machine. Resource cost is defined by the execution time and storage needed to conduct the computation.

Definition 1.1.2 (Software complexity)**.** The measure of the resources expended by

another system in interacting with a piece of software, where the interacting system consists of people. Software complexity is defined by the difficulty to perform tasks such as debugging, testing, or coding.

This thesis aims to answer the following research questions (RQs):

(**1**) What are suitable artifacts for optimizing ML models concerning resource cost/software complexity trade-off?

(**2**) Applied to an ML model, how successful are the artifacts in prioritizing the resource cost/software complexity trade-off?

The main deliverable of this thesis, which aims to answer RQ(**1**) and provide grounds for answering RQ(**2**) is a framework for ML developers, as mentioned above. A definition of a framework provided is [7], *A framework is a particular set of rules, ideas, or beliefs which you use in order to deal with problems or to decide what to do.* Given this definition, we choose to define the term framework referenced in this thesis as *The set of artifacts that are used in order to deal with problems or to decide what to do.*

Artifacts in this context are both quantitative metrics and qualitative assessments. To exemplify, there will be a metric for training- and prediction runtime analysis in the framework which is an absolute number that can be compared, apples-to-apples, between different models. There will also be assessments to compare, e.g., explainability and interpretability between models. This comparison is of a more qualitative nature and more cumbersome to compare and hence should not be called metric, but rather assessment. To be able to use one word for both to create clarity around the research questions, the term *artifact* will be used.

In RQ(**1**) the word 'suitable' refers to *acceptable or right for someone or something* [8] which in this context means that the artifacts should ideally be model unspecific and general enough to be able to be implemented by any given ML team. 'Optimizing' means *to make something as good as possible* [9] which refers to how one can create as good a model as possible using the different artifacts in the framework as means to optimize based upon, given use case-specific properties.

In RQ(**2**) the word 'prioritizing' is used, which means *to decide which of a group of things are the most important so that you can deal with them first* [10], which in this context relates to the optimization formulation in RQ(**1**) and the trade-off between different artifacts, depending on the requirements of the use case at hand.

Finally, 'trade-off' means *a situation in which you balance two opposing situations or qualities* [11], which refers to that it might not always be possible to optimize both resource cost artifacts and software complexity artifacts at the same time, depending on the use of case-specific requirements. The assumption that such trade-offs exist is fueled by, for example, that performance (mapped to software complexity) of an ML model is driven by the amount of data (mapped to resource cost) that is available. In addition, we know from our past experience that different use cases make different model types more or less relevant. For example, neural networks yield high performance but they are not inherently interpretable [12] and require rather large amounts of data. This thesis with RQ(**1**) aims to give a structured approach

for optimizing artifacts in both resource cost and software complexity, where this structure additionally should provide guidance towards what trade-offs might exist for a use case and how these trade-offs should be prioritized.

1.2 Outline

Chapter 2: Contains theory on the differences between ML and SE, a presentation of the ML development process referenced in the thesis paper, and software complexity and resource cost, with relevant subsections for both.

Chapter 3: Presents the methods used to research the RQs in the thesis, and how each part of the method contributed to the two RQs.

Chapter 4: Answers RQ(**1**) by presenting the framework generated from the material gathered in Chapter 2 together with the interview analysis in the first subsection of this chapter. Answers RQ(**2**) by presenting findings applied to an existing ML model at the partner company as well as the two alternative models that were developed for the same problem.

Chapter 5: In this chapter a discussion related to the RQs is presented together with the limitations of the thesis, with one section for each of the two RQs.

Chapter 6: In this chapter, possible threats of validity and strategies to avoid/ reduce them are presented.

Chapter 7: Completes the thesis with the conclusions drawn related to the two RQs and the purpose of the thesis. Finally, the chapter ends with a section on future research.

2

Theory

Machine Learning (ML) is considered a branch of the broader field of Artificial Intelligence (AI) [13], which refers to how to create systems or computers that evolve and learn through experience over time [14]. The growth of ML is driven by the increasing availability of data together with low-cost computation power and the evolution of new theories in the field [14]. The use cases of ML can be found in a variety of industries including healthcare, finance, marketing, education, etc. [14, 15]. To build ML solutions a variety of different skills are needed including statistics, mathematics, and Big Data [16].

Another well-established technical branch is the discipline of software engineering, which refers to the production of quality software applying engineering, mathematical, and scientific methodologies [17]. Humphrey [17] defines the software engineering process as all activities needed to build software given a set of user requirements.

The overall purpose of this chapter is to equip the reader with the appropriate theory to be able to digest the framework, presented in Chapter 4. Machine learning development approaches (Section 2.1) introduces some generalized process maps for ML development projects. This section is important as the framework is structured based on the phases of the development process chosen, presented in Figure 2.1. As the framework is supposed to contribute to SE4ML and is supposed to mix ML and SE best practices to tackle software complexity and resource cost, Section 2.2 is purposed to highlight the main differences between practices. Software complexity (Section 2.3) consists of motivation of why it is important to address this type of complexity, including subsections that are meant to introduce the aspects of software complexity that we have chosen to include. The same structure goes for the final section, Section 2.4, addressing resource cost instead of software complexity.

2.1 Machine learning development approaches

To justify the structure of the framework we will in this section present different types of ML model development approaches and conclude which one we will use to structure the framework. The choice is based on granularity and our understanding of what the ML development process looks like at the partner company. Amershi et al. [18] present a simplified view of the ML development process, see Figure 2.1. The *Model Requirements* step tackles which ML model approach suits the given problem [18]. *Data Collection* handles the data collection, *Data Cleaning* tackles

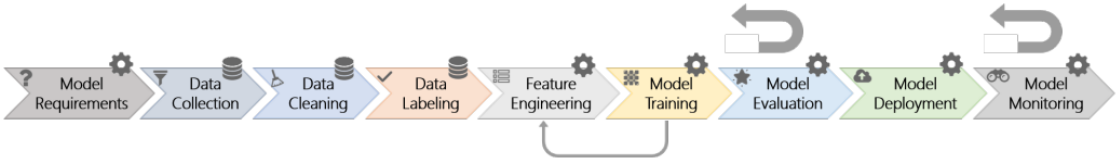

Figure 2.1: The ML development process presented by Amershi et al. [18].

removing noisy data, and *Data Labeling* takes care of labeling data whenever necessary, for instance in image recognition problems [18]. *Feature Engineering* refers to the extraction of useful features, *Model Training* handles the training and tuning of a model on the cleaned training data, *Model Evaluation* takes care of the evaluation of the trained model on the pre-defined metrics [18]. The two final steps are *Model Deployment* which is when the model is actually deployed in the system, and *Model Monitoring* which refers to the continuous monitoring of the model in the live environment. Hill et al. [19] describe the ML development process in the steps *Define problem, Collect data, Generate Ground Truth, Select Algorithm, Select Features, Generate and Evaluate model*. Hill et al. [19] highlight that the steps defined in their paper were not always performed in the exact sequence they are presented in and neither were all steps always performed by their interviewees. Ashmore, Calinescu, and Paterson [20] presents the ML model lifecycle which consists of four stages; *Data Management, ML, Model Verification, and Model Deployment*. In the first part of the lifecycle, Data Management, Ashmore, Calinescu, and Paterson [20] combine the work of obtaining data sets for training and verifying the models together with the phases of data collection and data preprocessing. In addition to the aforementioned ML development process and lifecycle, more general data science and data mining development processes include CRISP-DM (CRoss-Industry Standard Process) and KDD (Knowledge Discovery Databases) [16]. In this thesis, the ML development process defined by Amershi et al. [18] visualized in Figure 2.1 will be used as it maps better to the ML development context at the partner company, and it is also more granular which helps with making the framework artifacts easier to use.

2.2 Differences in machine learning and software engineering practices

The purpose of this section is to conclude which are the main differences between ML and SE engineering practices. This is important as some of these concepts will be used to justify some of the artifacts that will form the framework. The main takeaway of the section is three aspects in which ML practices differ from software engineering practices; (1) data discovery and management, (2) model customization and reuse, and (3) modularity [18]. Each subsection is supposed to contextualize these three main aspects, respectively.

2.2.1 Data discovery and management

While software engineering practices focus primarily on the code, ML practices take a more data-driven approach [18, 21]. Data handling and data preprocessing are in focus in ML practices, as the performance of an ML model is highly data-dependent. Wan et al. [21] states that it is thus of high importance in ML development to spend effort on data handling. The ML development process is also more experimentally driven than in traditional software engineering, as multiple ML algorithms can be candidate solutions.

2.2.2 Model customization and reusability

As for the second aspect mentioned by Amershi et al. [18], in traditional software engineering, it is common practice to reuse code in the forms of algorithms, functions, modules, etc. ML models are often built and trained for a given context, where model algorithms are chosen and parameters are set during the training of the model. This means that, according to Amershi et al. [18], one cannot just change parameter names if one wants to use the same model for another context. For a similar domain and input the model can rather easily be reused, but as soon as the context changes the model needs to be retrained and assessed on new sets of data, or in some cases complete model replacement is required.

2.2.3 Modularity

Lastly, Amershi et al. [18] claim that it is difficult to maintain ML module boundaries due to two reasons; low extensibility and non-obvious module interactions. For two or more ML models with different functionalities, simply adding them together will not create a properly functioning model. Instead, the models would have to be developed and trained together in order to function together. For a system consisting of two or more ML models, the models might interact in unexpected ways, impacting each other's training and tuning processes. Therefore, collaboration and training of the system as a whole are needed.

Wan et al. [21], however, disagree with the lack of modularity in ML and claim that ML models consist of relatively fixed modules in terms of data collection, data preprocessing, data modeling, and testing and have comparably low coupling to non-ML models. In this thesis, we claim that there exists a level of modularity for ML development, and therefore our interpretation of modularity is based on the reasoning of Wan et al. [21]. We make this claim as the ML team at the partner company uses modularity in accordance with this definition.

2.3 Software complexity

As presented in Chapter 1, we define software complexity as the difficulty to perform tasks such as debugging, testing, or coding if the interacting system consists of people [3]. The ambition of this section is to clarify which aspects of SE and ML complexity are embedded in the software complexity definition used in this thesis.

Already in 1989 Banker, Datar, and Zweig [22] wrote a paper on how software complexity influences software development costs. The authors argue that software complexity mainly influences the software maintenance labor costs as more complex software leads to more hours spent on maintaining the modules [22]. Canfora and Cimitile [23] highlight, with references from various studies, that software maintenance stands for between 60% to 80% of the total software life cycle cost. Software maintenance is defined as all activities performed on the software module after the software module has become operational [23, 22, 24]. Ogheneovo [24] shows how the increased size of software leads to increased maintenance costs where the main reason, the author argues, is an increased learning curve of the people that are to maintain the software, who need to spend more time understanding the software which in turn leads to a higher cost when maintaining it.

Given the aim to contribute to SE4ML, which means that we are in the interface between SE and ML, this section will cover understandability (2.3.1), ML model complexity (2.3.2), data preparation (2.3.3), entanglement, and data dependencies (2.3.4), and explainability and interpretability (2.3.5). The main takeaway is that these five subsections are considered to be the main areas connected to the definition of software complexity used in this thesis. Some of them originate from the ML research field while others originate from the SE research field, which explains the logic of the headlines. This means that they are the foundation of all artifacts that are connected to tackling software complexity, further presented in Chapter 4.

2.3.1 Understandability in a software engineering context

According to Trockman et al. [25] previous research shows that developers spend up to 70% of their time understanding code. Kaur et al. [26] argue that software complexity is dependent on the understandability of the code which means that programs are considered complex if the code is difficult to understand. The authors suggest that software complexity is an estimate of the number of efforts that are needed to develop, maintain, and understand the code [26]. Understandability is closely related to maintenance-related activities [25] and understandability can be considered as a subsection to maintainability in the Software Quality Characteristics Tree defined by Boehm, Brown, and Lipow [27]. According to the tree, understandability can be broken down further into self-descriptiveness, structuredness, conciseness, consistency, and legibility [27]. Trockman et al. [25] conclude that specific variables, fewer parameters, and fewer lines of code are associated with more readable code.

2.3.2 Machine learning model complexity

Hu et al. [28] conclude that ML model complexity has been investigated by different researchers for decades. There are many reasons why one might be interested in ML complexity; for debugging activities, gaining trust in the predictions, etc. [29].

Sculley et al. [30] argue that ML models both have normal code-level complexity, similar to the complexity found in software engineering modules, and larger system-level complexity. The system-level complexity refers to the interaction between the

ML models and the larger system, with the conclusion that hidden technical debt can accumulate in this interface [30]. Measures used to detect and assess ML model complexity are dependent on what type of model one wants to review.

Hu et al. [28] summarize how different ML techniques issue different ways of measuring their complexity. An example is decision tree models where complexity is normally represented by the tree depth of the model together with the number of leaf nodes [28].

2.3.3 Data preparation

The quality of the data is one of the largest success factors of any ML algorithm [31, 32]. We see data preparation as a factor/relevant aspect for software complexity as it is, in many cases, manual tasks that are conducted by people, and hence connected to how people interact with a piece of software. Data preparation consists of three main activities according to Chai et al. [31]; Data discovery, Data cleaning, and Data labeling.

Data discovery aims to retrieve and collect data from external sources such as the web, data warehouses, and data lakes. Data cleaning aims to clean up the retrieved data from noise which could be outliers, missing values, etc. The last step is data labeling, which aims to label the data by using different means such as crowdfunding or similar [31].

As presented in Section 2.1 Amershi et al. [18] have denoted the data preparation steps slightly different compared to Chai et al. [31]. However, the actions required in the different steps are similar which means that a practitioner needs to collect and understand the data, clean it by removing outliers, missing values, etc., and finally label the data if applicable. The labeling part is not always required, Amershi et al. [18] mention image recognition problems as an example of when this step is important, and more generally this is mainly required for supervised ML problems.

2.3.4 Entanglement and data dependencies

There are a number of studies that tackle the root causes of ML model system-level complexity, where entanglement and data dependencies are the most mentioned causes [18, 33, 34, 35]. Entanglement means that there are no inputs to ML models that will ever be entirely independent, as changing a hyperparameter or adding/ removing a feature will have an impact on other hyperparameters and features in turn [30]. Possible ways of mitigating entanglement are by isolating models or focusing on the detection of prediction behavior changes.

Data dependencies mean that the ML model performance is dependent on the data that is fed to the model. Sculley et al. [35] account for two types of data dependencies; unstable and underutilized. Unstable data dependencies refer to how data changes over time, due to distributional shifts in the data or if a model is dependent on another ML system that gets updated. These dependencies can be mitigated through versioning, i.e., that a version of the model is frozen until an updated version is carefully examined. Underutilized data dependencies refer to unneeded input

Table 2.1: Possible causes of underutilized data dependencies [35].

Cause	Description
Legacy Features	Features that remain from early development and are redundant by new features
Bundled Features	Group of features added to the model together that possibly include features of little to no value
ϵ-Features	Features added to improve accuracy although the gain is small and the overhead complexity might be large
Correlated Features	Strongly correlated features where only one is directly causal, where the ML model might choose the non-causal feature

that remains in the model. Table 2.1 displays different ways in which underutilized data dependencies can arise in a system. They can be detected through a leave-one-out analysis of the features [35]. As for correlated features, it is usually difficult to detect which feature in a pair of correlated features is the direct causal one. Causal inference is a technique that can be used to detect whether a feature actually has a causal effect on the target variable [36]. Spirtes [36] account for the causal Bayesian networks method for understanding causality, where one draws a directed acyclic graph (DAG) for a model. The DAG shows all relevant features and the edge direction between them shows how the features relate to one another. From this possible confounders can be detected, meaning features that impact both the feature of interest and the predicted outcome. Thereafter, possible confounders are controlled for when checking if a feature has a causal effect on the predicted outcome [36].

2.3.5 Explainability and interpretability

When applying ML algorithms to real world-problems, serving as decision-making or decision-making guidance for decisions usually entrusted to humans, there is a need for these algorithms to explain themselves [37, 38]. The terms explainability and interpretability are in some papers used interchangeably [38, 39, 40], while others claim a distinction between the two [12, 41].

Nauta et al. [40] motivate having a wider definition of explainability by creating a more inclusive discussion and including more sources and define explainability as *A presentation of (aspects of) the reasoning, functioning and/or behavior of a machine learning model in human-understandable terms.*

Rudin [12] on the other hand claims that interpretability refers to designing/choosing inherently interpretable models, while explainability refers to post hoc explanation of black-box models such as neural networks [39]. In this thesis, we chose the same definition of interpretability as Rudin [12], and explainability as post hoc explanations of models. Post hoc in this context means explanations provided after the model is fit to the training data.

The need for explainability and interpretability. The need for explainability

and interpretability is motivated by several reasons. Lipton [42] emphasize the need for explanations behind critical decision-making ML applications such as, e.g., loan applications and hiring tools.

Rudin et al. [41] highlight the need for interpretable ML models in case of a domain shift, i.e., that the input data distribution changes over time, as troubleshooting, will be easier for more interpretable models which in turn will lead to better accuracy. The authors [41] also mention the use of interpretable ML in the full development process, as tuning of model parameters and processing of data benefits from interpretability.

However, as explainability and interpretability are emphasized in many contexts, it is also less necessary in others [41, 42]. Examples of situations as such are low-stake decisions such as advertising, trivial decision making, and any time humans easily can verify and modify the decision of the model.

Evaluation of explainability and interpretability. To evaluate the explainability of a modeling process, one can examine interpretable models and their attributes, as well as explanation methods [39].

Rudin et al. [41] highlight that explanation techniques are not as reliable as inherently interpretable models, and, thus, advocate using inherently interpretable models for conclusions rather than solely depending on post-hoc explanation techniques. Examples of more interpretable models mentioned in the literature are sparse logical models (such as decision tree models) and generalized additive models (GAMs) [39, 12]. As interpretable models are naturally dependent on model choice [39, 12], this section will only account for explanation methods in further detail.

Explanation methods can be divided into local and global explanation methods, where local methods help explain single instance predictions, while global methods explain decision-making for the overall structure [39, 40, 43]. Another division of explanation methods mentioned by Marcinkevičs and Vogt [39] is between model-agnostic and model-specific methods. Model-agnostic methods are applicable to any ML model, while model-specific methods are applicable to a restricted group of ML models [39].

Examples of local, model-agnostic methods are Shapley additive explanations (SHAP) and Local Interpretable Model-agnostic Explanations (LIME). SHAP shows feature impact by comparing the impact of a feature value compared to a baseline value and explains a prediction as to the sum of feature value effects [39]. LIME tweaks feature values for a single sample and analyze the impact on the output [39, 40].

Examples of global, model-agnostic explanation methods are Partial Dependency Plots (PDPs) [39, 40, 43] and Individual Conditional Expectation (ICE) [44]. Both methods visualize the interaction between features of interest and target response, under the assumption that the features of interest are independent of the complementary features [45].

2.4 Resource costs

This section aims to introduce the term *resource cost* as defined in Chapter 1 as the execution time and storage needed to conduct the computation if the interacting system is another machine [3]. Important to notice is that *cost* can be of any unit, not only monetary units. Sections 2.4.1–2.4.2 will present ways of measuring resource cost in the context of this thesis. The main takeaway is that these two subsections are considered to be the main areas connected to the definition of resource cost used in this thesis. This means that artifacts connected to resource cost in the framework will originate from the concepts presented in this section.

2.4.1 Data volume and Google BigQuery introduction

According to Chai et al. [46] more traditional ML algorithms tends to plateau in terms of performance as training data size increases whereas more advanced deep learning models tend to have an increased performance. Having a sufficient amount of data given the needs of the model in development is paramount for creating business value [46]. Depending on the company and its specific setup, different data storage platforms can be used.

The partner company in this thesis is using Google Big Query as the interface between the Google Cloud Platform and its applications. BigQuery is a scalable and fast product provided by Google with in-built ML capabilities as part of the product [47]. BigQuery is a serverless product which means that the data is physically managed by Google's servers and accessed through the cloud in an on-demand manner [47].

The cost model of the service is dependent on how much data is stored and used on the platform, meaning that costs increase linearly with usage [48]. The cost model is divided into how much data is stored and the amount of data computed [48]. The storage pricing depends on the amount of data stored, priced at pennies per gigabyte of data stored, with half of the price for long-term storage (data not modified in the last 90 days) [48]. The computed data is charged by bytes of data processed in $ per terabyte, which is not the same as the number of bytes returned to the user but rather the computing power needed to return the data [48]. There is also a flat rate available where the user gets a number of slots used to execute the queries, and the user pays a monthly fee to cover these fixed amounts of slots [48].

2.4.2 Training and prediction time

The last aspect related to resource cost that this thesis aims to include are training and prediction time. These metrics are, given the ML context of this thesis, functioning as execution time in the definition for resource cost presented in Section 1.

Lim, Loh, and Shih [49] use training time as one of three measurements used to compare the performance of different models (together with accuracy and tree-depth). They argue that despite the fact that implementation techniques can affect the training time, one will still be able to use training time to compare models. Especially

if one witnesses differences of, e.g., several minutes comparing models, a difference most likely not linked to the implementation technique [49]. Also, Das and Behera [50] use training time as a parameter in their paper aimed to explain and compare some popular ML algorithms. In addition, they also include prediction time as a standalone metric in their assessment [50].

3
Methods

The method in this thesis consists of a mixture of literature review (Section 3.1), unstructured and semi-structured interviews (Section 3.2), and ML model experiment (Section 3.3). Table 3.1 displays how each method step maps to the research questions of this thesis. The research questions defined in Chapter 1 are:

(**1**) What are suitable artifacts for optimizing ML models concerning resource cost/software complexity trade-off?

(**2**) Applied to an ML model, how successful are the artifacts in prioritizing the resource cost/software complexity trade-off?

The first steps of the research methodology in this research can be described as information gathering from literature mixed with input from the unstructured discovery interviews conducted at the partner company. This output is then analyzed and formalized into a framework, addressing RQ(**1**). The output of the unstructured discovery interviews was also used to prepare for addressing RQ(**2**), giving an understanding of the experiment context.

The framework focuses on the ML model development process described in Section 2.1, and contains artifacts used to assess the software complexity and resource costs, defined in the introduction (Chapter 1), in every phase of the development process. Thereafter, portions of the framework are tested in a more experimental setting on an ML model provided by the partner company that is tweaked, as well as on two other ML models applied to the same problem to address RQ(**2**). The purpose of RQ(**2**) was to provide guidance on how we evaluated different models using the artifacts to justify and concretize how we interpret outputs from the various artifacts. In parallel with the ML model experiment, the framework was validated with stakeholders at the partner company together with independent researchers. This was done to add another layer of validation of the defined framework, referred to as *framework validation* in this thesis, addressing RQ(**1**) as it was conducted as a

Table 3.1: Mapping of RQs to method steps conducted.

Method Step	**RQ1**	**RQ2**
Literature review	X	
Unstructured discovery interviews	X	X
Semi-structured framework validation interviews	X	
Experiment		X

means to generalize the framework. Lastly, a discussion and conclusion on the trade-off between the software complexity and resource cost artifacts given the results of both RQ(**1**) and RQ(**2**) are given in Chapter 5 and Chapter 7 respectively.

The overarching method applied in this thesis can be mapped to Quadrant 1 in the ABC framework displayed in Figure 3.1, research conducted in natural settings. This given that the research of this thesis is conducted using the partner company's context. This chapter is divided into the sections of literature review, interviews, and ML model experiment, where the first two maps to the *Field Studies* section in Quadrant 1, see Figure 3.1. A field study is research conducted in a real-world setting, where the researcher does not intrude on the setting [51]. The literature review conducted is to be seen as theoretical support for the results of the interviews. The last step of the method, the ML model experiment, instead maps to the *Field Experiments* section, also found in Quadrant 1 (Figure 3.1). A field experiment is just like a field study conducted in a real-world setting, where the researcher manipulates some properties in the setting to observe an effect [51]. The ML model experiment manipulates the setting by applying the framework to an existing problem at the partner company, where the observed effect stands for the outcomes of applying the framework.

3.1 Literature review

A literature review was conducted to explore current research within the scope of our thesis, mainly related to finding relevant artifacts as part of RQ(**1**). The literature review conducted in this paper follows the five research stages formulated by Cooper [52]. The literature search was done using mainly Google Scholar as a database, where the initial search terms were based on the scope of the project including terms such as *machine learning model complexity* and *software complexity measures*. More examples of search terms can be found in Table 3.2.

Further, in the literature search, the search terms were based on titles and content of previous relevant search results. Additional sources were found by snowballing the results of the literature search. The immediate results were chosen based on a quick relevance scan, looking at a paper's title, abstract, and conclusion. Thereafter, the results were filtered on scope relevance after further data evaluation by reading them as a whole. Additionally, a portion of the literature used was provided from recommendations from our university supervisor as well as the ML team at the partner company.

3.2 Interviews

Two types of interviews were conducted in the course of this thesis. The first set of interviews, referred to as *unstructured discovery interviews*, are described in Section 3.2.1 and aim to address RQ(**1**) and to understand the context in which the partner company operates to provide material for the framework. We describe the context of the partner company and which teams we interviewed, how we conducted

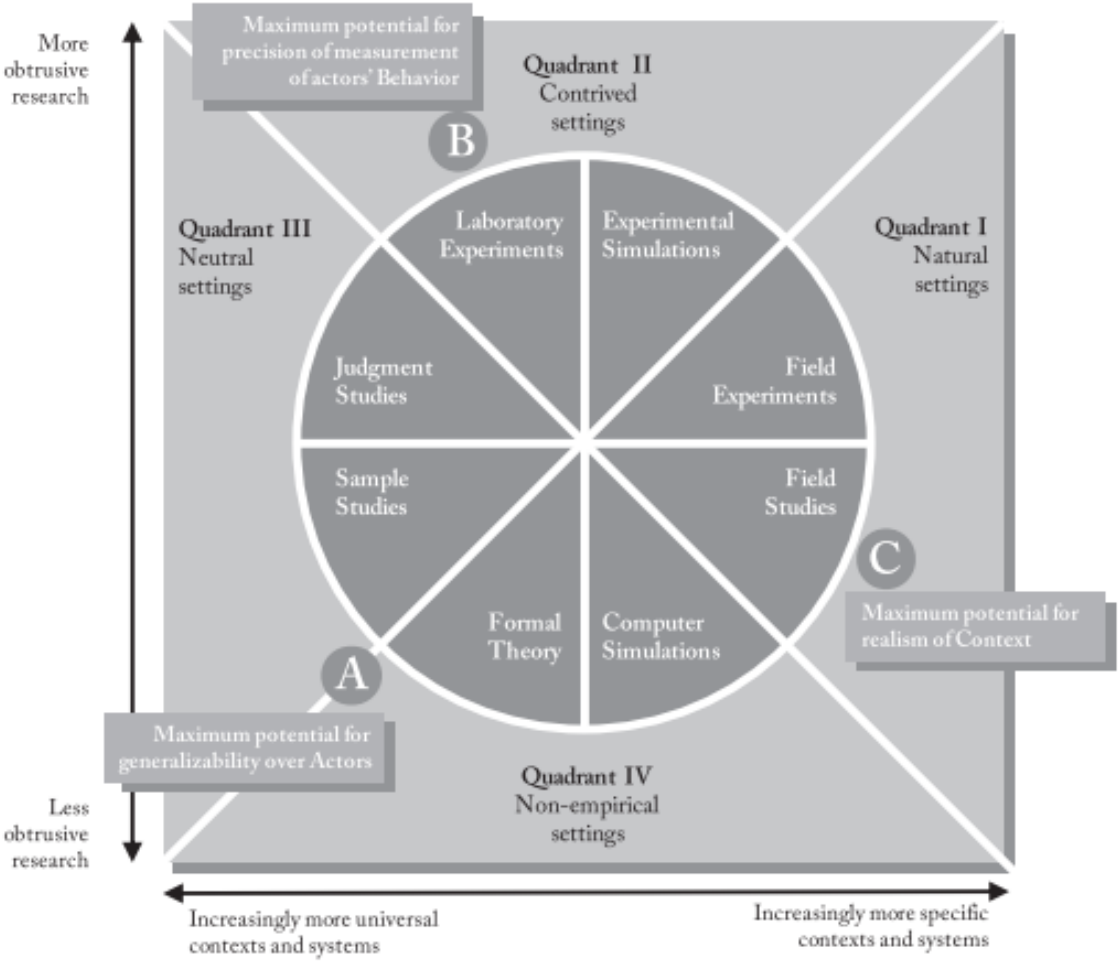

Figure 3.1: The ABC-framework by Stol and Fitzgerald [51].

Table 3.2: Examples of literature search terms

Search terms
Machine learning model complexity
Software engineering complexity
Machine learning complexity measures
Software complexity measures
Software development costs
Non-functional requirements

the interviews, how we analyzed the outcome, and finally, how we presented the outcome as part of the result.

The second set of interviews are referred to as *semi-structured framework validation interviews* and are also meant to address RQ(**1**) by interviewing experts in the field, and stakeholders of the partner company, to validate the proposed framework. The deductive approach that was used for the semi-structured interviews can be found in Section 3.2.1.

3.2.1 Unstructured discovery interviews

Unstructured discovery interview context. The interviews were conducted with different teams at the partner company, both technically oriented teams within ML, SE, and Data Engineering as well as with Commercial stakeholders. The Commercial team can be described as a team interfacing the customers of the partner company. Their responsibility is to drive initiatives that increase the happiness of the customers to drive more revenue and a better customer offering. The engineering teams can be described as teams working with data management, web, and app development. They are stakeholders of the ML team in the sense that they either deliver data to the ML models to run on or that they receive predictions or other deliverables from the ML team that they embed in their artifacts. They are referenced as software engineers in the context of this thesis. The ML team at the partner company consists of three people, one product owner overseeing the work and driving the communication with other teams, and two machine learning engineers.

Unstructured discovery interview approach. According to Hove and Anda [53], interviews are a good tool to obtain insights from individuals that are of qualitative nature. In this study, which has been conducted in close collaboration with a partner company, the interviews and meetings with various stakeholders have been essential in gaining insights used to form the outcome. In the early stages of the study, we conducted discovery calls with various stakeholders from Commercial, Engineering, and ML teams, with questions such as “What is your main scope of work?”, “How do you interact with other teams?”, and “What use cases do you have that are connected to ML models?”

The interviews were 30 minutes long and conducted with 1–2 representatives from the 4 teams that were interviewed. The interviews were not recorded. Instead, one of us focused on asking questions while the other focused on writing notes. In the study by Hove and Anda [53] an analysis was presented showing that it might be beneficial to be two researchers conducting the interviews as it may increase the interview length by up to 60%, assuming that all input of an interview is useful. In addition, the interpretation of the interviews may be more accurate [53].

Given that the partner company is based in Sweden, with Swedish as the corporate language, and the fact that both researchers are Swedish, all interviews and notes were held and written in Swedish. The outcome of the interviews, presented in Section 4.1, was translated from Swedish to English and the interpretation of the outcome was validated by both researchers to ensure correctness.

These first rounds of discovery interviews could be described as unstructured, given

Table 3.3: A list of the interviewees with whom the discovery interviews were conducted. The first column indicates if the representative was alone in the discovery interview or accompanied by a colleague. Experience is abbreviated as 'exp.' for readability purposes.

#	Role	Team	Years of exp.
1	Product Owner	Data Management	10
2	Technical Project Manager	Business Development	8
2	Product Owner	Sales Data Insights	6
3	Software Developer	Product Development	3.5
3	Software Developer	Product Development	10
4	Machine Learning Developer	Data	2
4	Machine Learning Developer	Data	0.5

the open setting and that we had no instrument with questions prepared in advance. Instead, we let the interviewees speak openly about their position, interaction with the ML team, etc. The rationale for applying this unstructured approach was that the interviews were conducted early in the research process. Hence, we wanted to understand the context of the partner company from different perspectives. An overview of all interviewees in this discovery phase can be found in Table 3.3.

We analyzed the main takeaways from the interviews and compared them with the theory chapter in every specific section to see whether or not the interview contents were aligned with the literature. We highlighted the main similarities and differences from the literature and finally, we created distinct subsections based on the analysis (see Section 4.1).

Semi-structured framework validation interviews. As a final step, aimed to validate the proposed framework, so-called validation interviews were conducted. The purpose of these interviews was to test whether the framework made sense both to practitioners at the partner company and to representatives from academia. The analysis process of the interviews can be described as deductive [54], as the interviews were conducted to investigate a hypothesis, in our case the framework.

In addition to the ML team at the partner company, four independent researchers were selected based on input from the thesis supervisor. All of them had between 15-25 years of research experience, mainly from SE but with at least 4 years of ML experience. After being invited to an interview slot all interviewees were equipped with a pre-read version of the report containing the introduction to the thesis and an explanation of the purpose together with the framework and explanations of all included artifacts. During the interviews, all of which were conducted online, one of us drove the conversation and asked the questions while the other one took notes and asked complementary questions. After consent from the interviewees, all interviews were also recorded as a backup to the notes to avoid misinterpretations. All questions were structured and asked in the same order to all interviewees.

The first questions were closed-form, to gather data about the interviewees which is

Table 3.4: A list of the interviewees that were interviewed as part of the framework validation process. In total 6 people were interviewed with various degrees of experience within ML and SE, as can be seen by the statistics in the table. Experience is abbreviated as 'exp.' for readability purposes.

#	**Industry exp.**	**Research exp.**	**SE exp.**	**ML exp.**
1	1.5 years	18 years	18 years	4-5 years
2	1 year	15 years	16 years	15 years
3	3 years	22 years	25 years	7 years
4	3 years	16 years	16 years	4-5 years
5	2 years	0 years	0 years	2 years
6	0.5 years	0 years	0 years	0.5 years

presented in Table 3.4, while the questions relating to the framework were open to allow for follow-up questions, see Appendix A.1. The analysis of the interviews was conducted by coding the notes one-by-one to highlight items that appeared in the different interviews. The findings were then discussed and, reaching saturation, we arrived at a final subset of changes to the framework and the artifacts. Given the deductive approach, the main outcome of the interviews was mainly clarifications related to specific artifacts.

3.3 Experiment approach

This section includes a description of the steps taken during the experiment phase of this thesis, together with the work structure set up to obtain a structured and clear process and result. The experiment took place in order to address RQ(**2**). The main objectives of this method step were:

- To get a sense of the quality of the framework in practice
- To give examples of how the framework should be applied and what output to expect
- To use it in order to make a decision between three different candidate models

This section begins by walking through the development steps taken (Section 3.3.1), to provide an overview of the development process. Thereafter, an overview of the development structure is given (Section 3.3.2).

3.3.1 Development steps

The development steps taken were to a large extent following the steps in the framework presented in Section 4.2. The overall steps taken during the experiment can be summarized as:

1. Getting familiar with the problem, existing model, and GitHub repository
2. Implementing two alternative models

3. Data assessment, in this case, a review of the assessment conducted by the ML team
4. Model training, feature engineering, and runtime analysis done iteratively and in parallel
5. Preparation for handover and handover to ML-team

The modeling portion of this thesis is based on an existing model in the partner company's current ML GitHub repository. The model algorithm used is XGBoost [55], with the aim to predict the number of magazines to deliver to re-sellers using time series data.

The initial step, step 1, in the modeling method was to first get familiar with the repository, to then be able to adapt the code to a new repository, where features, models, and predictions were saved locally.

For comparison purposes and to find potentially better candidates for the problem, in Step 2 two more models were implemented; a basic linear regression model using Scikit-learn [56] and a Linear Generalized Additive Model (Linear GAM) using pyGAM [57]. These two models were implemented due to their inherent interpretability.

In the third step, we reviewed the data assessment conducted by the ML team at the partner company. The fourth step of the experiment consists of multiple analysis steps taken, namely the steps of feature engineering, training and evaluation. For interpretability and feature evaluation purposes, inherent methods of the three model implementations were used. These consisted of feature importance plots for the XGBoost, feature coefficients for the Linear Regression (LR) model as well as a model summary including relevant statistics for the Linear GAM.

For further explainability and feature evaluation purposes (see Section 2.3.5), to address the artifacts relating to the steps of machine learning requirements and feature engineering in the framework, SHAP plots were created for the XGBoost and linear regression models. The linear GAM library used was not supported for the SHAP plot. Additionally, PDP plots were plotted for the XGBoost model and the linear GAM. As in LR features are modeled to have a linear relationship with the output, PDP plots were thought to be excessive for plotting the feature coefficients. In short, we implemented all inherent methods for interpretability for the models, and the explanation techniques mentioned in Section 2.3.5 that a) were compatible with each model library and b) were deemed to possibly add further insight. In order to evaluate the training and prediction runtime for each of the models, the Python time model was used [58] and implemented for methods of interest. Lastly, the code was cleaned up and double-checked and then a walk-through with the ML team was conducted.

3.3.2 Development structure

Throughout the modeling process, both pair programming and solo programming were used. Pair programming is a software development approach where two team members work with the code at the same time, using one computer [59], an ap-

proach proven to generally yield better results and happier programmers according to Williams et al. [59].

Due to time constraints, the pair programming was complemented with solo programming, where the team member with the most machine learning development experience was mainly assigned to the programming tasks at hand. The code is written in Python, wherefore the PEP-8 Python coding standards [60] were followed.

4

Results

This chapter consists of four parts; *Unstructured interview analysis* (Section 4.1), which aims to map the outcomes from the interviews with the theory in Chapter 2. *Framework presentation connected to RQ*(**1**) (Section 4.2) aims to introduce the framework and describe all artifacts that it consists of. *Validation interview analysis* (Section 4.3) which give an overview of the feedback received on the framework during the validation interviews, and finally, *ML model experiment connected to RQ*(**2**) (Section 4.4), which aims to showcase how the framework can be used throughout the ML development process.

4.1 Unstructured discovery interview analysis

This section presents the relevant outcomes from the interviews with the ML, Engineering, and Commercial teams at the partner company. The section aims to address RQ(**1**), by contextualizing the theory presented in Chapter 2. The interviewed teams and their areas of responsibility are described in Section 3.2.1.

Section 4.1.1 aims to justify using the development process presented by Amershi et al. [18] in the framework by comparing it with the process at the partner company. Section 4.1.2 focuses on contextualizing modularity and reusability using the partner company as a reference case. Section 4.1.3 gives an overview of the partner company's data management, and finally, Section 4.1.4 links the partner company's deployment process to the theory.

4.1.1 ML development process justification

The ML team currently has no formalized ML development process that they follow. However, when the ML team is asked to describe the high-level problem-solving approach, they do follow some of the steps visualized in Figure 2.1 originating from the paper by Amershi et al. [18]. The team does not have a generalized process map in place of how the development process looks like, which is why we chose to map their process to the most similar one found in the literature.

Hill et al. [19] mentions that all interviewees in their study did not follow all steps of the process presented in their paper and neither the exact sequence of the process steps, which is too the case at the partner company. The conclusion drawn from this is that the ML development process, as defined in Figure 2.1, can be used to map the artifacts and activities to different parts of the process as a way to generalize it

to any given ML development process. The rationale for using this specific process (Figure 2.1) is because it maps the best to the high-level problem-solving approach shared by the partner company, regardless of whether all steps are completed or only a subset of them. Our assumption is that this process generally maps well with the modus operandi of other companies too.

> The outcome of this section maps to RQ(**1**) as it justifies using the ML development process by Amershi et al. [18] to map artifacts.

4.1.2 Modularity and reusability

The ML team reports that they have built a general structure of files in their development repository, that they use for all their different models. The structure facilitates that the appropriate environment variables are set at runtime, that the link to the data warehouse is set up correctly, and that there is functional modularity of the repository.

Elaborating on the latter, there are separate files for separate functionalities, where examples are one file for feature calculation and one for querying the data. The structure also consists of model inheritance, meaning that there exists a base version model of a certain modeling technique, and an instance of this technique can be created and further developed for the specific task at hand. As the team describes it as common to use the same modeling technique for different applications, they find using model inheritance useful in their work structure. Moreover, model inheritance and modularity help the team to focus on model development rather than basic code generation and infrastructure, which decreases the total development time needed. This practice can be mapped to the modularity and reusability discussions in Section 2.2 with contributions from Amershi et al. [18] and Wan et al. [21].

The former means that it is more difficult to reuse ML modules as the context is important for ML modules to function properly, which seems logical for the parts of the ML code that relates to the algorithm and training of it. However, in the case of the partner company, it seems like they apply modularity similar to the arguments presented by Wan et al. [21] where they state that the fixed modules related to data collection, data preprocessing, etc. have low coupling and hence can be reused.

> The main outcome of this section is the contextualization of modularity and reusability using the partner company as the reference case. This outcome will be used when describing later artifacts, as part of RQ(**1**).

4.1.3 Data management

The partner company uses BigQuery as an interface for retrieving the data from a live updated Google Cloud Platform. The data engineering team reports that they use the BigQuery on-demand service, meaning that the partner company is billed based on the data computed and stored in a given month. This means that every

time a model queries data, there is a cost associated with that specific query. More context on how this works can be found in Section 2.4.1.

The data engineering representative reports that this is sometimes perceived as a limitation and that they have set up quotas on the majority of the projects to guard the costs. The ML team reports that they have logs highlighting the data cost every time a model is run in Python, which makes them informed about the cost of that specific query.

However, simply using less of the data that one is querying when building models will not make the cost lower as it is the total cost of performing the query, and not the amount of data that is fed back to the user, that drives the cost. Additionally, the ML team mentioned how for, e.g., the time series problem used in the experiment phase in Section 4.4, the large data volume becomes an issue due to long training runtimes.

The outcome of this section is to contextualize how data management works at the partner company to interpret data volume. This outcome will be used as a part of the artifacts in the framework, linking to RQ(**1**).

4.1.4 Model deployment process and understandability

The process of deploying the model into production is not formalized at the partner company. Depending on the use case the deployment of the model may be done in various ways, the ML team reports. There is currently no model that is interacting with users live, neither in internal systems nor externally towards users and customers. In the current models that are live in production, the model outputs are either shared via API calls, shared via a file upload, or data tables that are uploaded to the data warehouse. This is relevant as it addresses the latter part of the ML development process in Figure 2.1.

The team reports that there is no formalized process for transferring knowledge of the model or even for documenting the model for other team members to review for future use cases. Instead, the team works with mob programming on a frequent basis in a rather informal manner. Mob programming is a development approach where the full team works on the same things, at the same time, using one computer [61].

As indicated by the commercial team, the model is being explained upon request which, according to all parties, works rather well for the moment but given what is being presented by Ogheneovo [24] in Section 2.3, where increased maintenance costs are driven by increased learning curves of developers that are tasked to understand the code in order to maintain it, this approach may be turned into a risk as to the team size increases. Moreover, Kaur et al. [26] is being referenced in Section 2.3.1 arguing that understandability drives code complexity from a SE standpoint which we mean also holds true for ML artifacts, as part of the code-level ML model complexity mentioned by Sculley et al. [30] in Section 2.3.2.

The main outcome of this section is how the deployment process works at the partner company linking it to the theory. This discussion too maps into the artifacts of the framework, related to RQ(**1**).

4.2 Framework presentation connected to RQ(1)

In this section, we present the developed framework, which is the main deliverable related to RQ(**1**). The purpose of the framework is to act as guidance for how software complexity and resource cost, defined in Chapter 1, may be addressed in different parts of the development process by the ML developers. This aims to contribute to the field of SE4ML by applying software engineering and machine learning practices to an ML context, which means that the artifacts in the framework originate from both SE and ML. The framework also aims to help the practitioner optimize the trade-offs between resource cost and software complexity artifacts. Ultimately, the optimization of the trade-offs will always be dependent on the specific use case, which is why the framework does not directly handle potential trade-offs. Rather so, the framework will give guidance on how to optimize the different artifacts in order to simplify the trade-off discussion. To further contextualize this trade-off discussion, we in Chapter 5 provide a discussion on possible trade-offs, to a large extent mapping to the experiment conducted in Section 4.4.

A visualization of the framework mapped to the different ML development phases, as presented in Figure 2.1, can be found in Figure 4.1. We have mapped the artifacts of the framework to the different phases to visualize in which order they should take place, according to our findings.

The main outcome of the framework is the different artifacts that act as guidance for ML developers when building new models. Given that the artifacts are mapped to the generalized development process, we believe that they will be general enough to guide ML development teams in many industries for different use cases.

Finally, to clearly visualize the input, output, and purpose of each artifact we have prepared Figure 4.2 to be used as a checklist for any ML practitioner.

The framework was built around the ML development process presented by Amershi et al. [18], visualized in Figure 2.1. The reason for this was a) to make the framework easy to implement in the current work structure of most ML engineers, b) to make it clear that throughout the development process different activities are important, and c) to structure certain activities to simplify the analysis of various models and the communication with stakeholders. As indicated in Section 4.1.1, this development process is also the closest to mapping into how the ML team at the partner company operates, which is another argument to map artifacts to it as part of the framework.

Some activities, for instance, whether the model needs to be interpretable given the use case, must be addressed early in the development process as it will influence which types of ML models might be beneficial (see Section 4.2.1). Other activities, such as feature analysis and importance will not be possible to assess until the design stage, and might not as clearly fall into one specific step in the development process

(see Section 4.2.3).

Due to the framework being designed in accordance with the ML development process, we suggest that ML engineers use the framework iteratively throughout the development process as a checklist to guide the creation and evaluation of different model candidates.

4.2.1 Machine learning requirements

In this section, all artifacts that are supposed to be assessed in the model requirements phase of the ML development process are presented. Due to the ambiguity of the word 'model', coming from one of the validation interviews where an interviewee in the field of requirements engineering thought that 'model' for them did not explicitly mean ML models, we choose to instead define the step as *Machine Learning Requirements* instead of model requirements.

In this phase, it is assessed which ML model approach suits a given problem, which is what the artifacts included in this section are supposed to support in answering. The main objective in this phase is to align the expectations of the model and to define what is important given the use case at hand. For example, if you have a use case that requires a highly explainable and interpretable model, one might need to move away from more complex and potentially higher-performing models to satisfy this requirement, which is likely to create a trade-off discussion. We mean that this is helpful to know early on in the development process to ensure good work efficiency throughout the development process.

As it is likely that this will be a discussion in a group of ML engineers together with the main stakeholders, all artifacts presented in this part of the framework are of a qualitative nature.

MR1: Interpretability needs. Interpretability relates to software complexity as interpretability needs to be assessed by people interacting with a piece of software, and needs to be considered on a case-by-case basis. It is important to address what level of interpretability will be needed for the case at hand early in the model development process to avoid redundant development work later on. To exemplify, a linear regression model has a high level of interpretability given the few parameters used and linear relationships modeled between features and the target variable. In contrast, a neural network is not interpretable due to the complex architecture with neurons, activation functions, and layers.

To address which level of interpretability that will be needed in the model it is important to analyze the use cases together with key stakeholders of the project to understand where the model is supposed to end up and to what level the outcome of the model needs to be controlled by humans. Moreover, the objective of the discussions should be to potentially disqualify some model types if the need for interpretability is high. In Section 2.3.5 Rudin et al. [41], among others, highlight when interpretability is important and when it is not as important.

Software complexity / *Resource cost*	Machine Learning Requirements	Data Collection	Data Cleaning	Data Labeling	Feature Engineering	Training	Evaluation	Deployment	Model Monitoring
Artifacts	**MR1:** Interpretability Needs **MR2:** Performance Needs **MR3:** Modularity & Reusability Assessment	**D1:** Data Assessment *- Data discovery* *- Data cleaning* *- Data labeling (if applicable)*			**FE1:** Feature Analysis **FE2:** Data Dependency Analysis	**T1:** Data Volume	**E1:** Training & Prediction Runtime Analysis	**DE1:** Understandability **DE2:** Deployment Stakeholder Understandability	Out of scope

Figure 4.1: A visualization of the framework is presented in this section. Artifacts in red text are mapped as resource cost and normal black text highlights artifacts that are categorized as software complexity.

ID	Artifact	Input	Output	Why
MR1	Interpretability Needs	How important is interpretability given the use case at hand?	The degree of interpretability - output format to vary depending on use case	To disqualify models based on whether they are inherently interpretable or not
MR2	Performance Needs	How should performance of the model be assessed? What is considered being an appropriate performance level?	Alignment on how to measure and interpret performance of the model/-s developed in the project	To set expectations towards the stakeholders of the project in terms of what is reasonable to expect of the models To enable greater understandability of the finalized model
MR3	Modularity & Reusability Assessment	Which already existing modules in the code stack can be reused for the solution? Which portion(s) of the new development could be reused/ purposeful to keep modular for future development?	A map of which modules to reuse when coding and what potential adaptations will be needed What modularity to try and implement throughout the development process	To plan what, in terms of coding efforts, needs to be done as part of the project to mitigate redundant work
D1	Data Assessment	What quantity of data is available? What is the availability of data? How does the structure of the data look like, e.g. in which tables are the data located? How easy is it for the ML team to access the data? What cleaning efforts will be needed to have the data ready for the task? Which labeling efforts needs to be conducted in order for the data to be ready to be used?	A complete overview of the answers to the questions in order to take this into account when building the ML model.	To understand and to be able to take into account how much data is available for the given project, to get a sense of the level of quality of the data, and to detect and possibly mitigate data dependencies
FE1	Feature Analysis	What features are considered to be useful for the specific problem at hand? What is the quality of the implemented features? What is the seemed importance to the model performance of each feature?	Feature check (e.g. feature distribution, missing values handling) Feature analysis through internal model techniques and explanation techniques (e.g. PDP plots)	To thoroughly assess all features to understand how the features behave, in order to potentially remove or to tweak some to operate more efficiently and possibly improve performance
FE2	Data Dependency Analysis	Does the feature set include any redundant features remaining from early development (legacy features)? Does the feature set include any bundled features? Does the feature set include any features that have a low impact on performance, in comparison to the added complexity (e-features)? Does the feature set include any pair of correlated features where only one of them is directly causal?	Potential removal of underutilized features by intuition and leave-one-out analysis Confidence in the fact that the features adopted by the model add value to the model	As data dependencies being one of the main ML model complexities there is a great benefit in taking the time to assess dependencies between different features
T1	Data Volume	How much data is needed in order to achieve reasonable performance? How does the amount of data needed to produce accurate predictions change for different models?	An analysis of the amount of data needed for different models to achieve a reasonable performance	Data greediness of models is important to take into account when scaling existing models as well as when models are live in production and monitored
E2	Training- and Prediction Runtime Analysis	How much time does it take to train the model? How much time does it take to generate predictions, using the model?	An apples-to-apples timedelta comparison between different models to be used as means to potentially disqualify models	To have a straight-forward concrete metric to be factored in when assessing multiple models
DE1	Understandability	How well documented and structured is the code currently? What can be done to make the code more understandable?	A plan on what code documentation exercises needs to be done to make the code easier to understand	As much as 70% of the time spent by developers are associated with understanding code, implying that this activity is important in order to improve maintainability, among other things
DE2	Deployment Stakeholder Understandability	What information would the main stakeholders require to feel confident about the usability of the model?	A document with information about the model including purpose, features, limitations, etc.	To facilitate effective communication between the developer team and the main stakeholders to ensure optimal usability and understandability

Figure 4.2: A visualization of which input each artifact requires and also what type of information each artifact is meant to output. The last column is meant to clearly express why a practitioner should spend time doing the artifact.

The main outcome of the MR1 artifact is to decide to what degree the model needs to be interpretable to guide the model selection process.

MR2: Performance needs. This artifact is mapped to software complexity as it relates to people interacting with a piece of software, as the artifact aims to decide the appropriate performance objectives for the models given a unique use case. In this context, 'performance' is defined as *how good a given model is at predicting the outcome*. For different use cases, different performance needs will be desired.

It is important to assess the appropriate performance level together with the stakeholders. Not only will this assessment help create a direction for which model types might be feasible for the use cases at hand but it will also help to create expectations towards various stakeholders of what is reasonable to expect from the model in terms of performance. Furthermore, it will probably make the expectations management towards stakeholders easier later on in the development process. The ML team at the partner company also mentions that it could be good to in this step educate the non-technical stakeholders on how one can assess a given model, for instance with MAE (Mean Absolute Error), to explain what such model-specific metrics are and what conclusions can be drawn from them. Finally, in the model evaluation phase of the ML development process Amershi et al. [18] mention that one should evaluate the model given pre-defined metrics, which according to the authors of this thesis, should be created together with the stakeholders in this assessment.

For a smooth evaluation process, we suggest that one creates some sort of baseline to compare to in this step. In the case that there is an existing model, this can be used as a comparison to the resulting metrics. This baseline could also be an evaluation of human performance for, e.g., a classification task. To decide on both suitable baseline and sufficient performance targets, one has to consider questions such as "do we need to outperform humans?", "should this model be used as a complement to human labor?", "how critical are false predictions?". E.g., in the case of a model that needs to replace human labor and where false predictions are severely damaging, a high-performance level is required. If it is instead a more trivial application acting as support to humans, a lower performance level might be acceptable.

In terms of performance needs, it was brought up by the ML team at the partner company that the smoothness of implementation of a model also relates to performance needs. An example of a smoother implementation is a model that requires only a Python library import with built-in functions for fitting and predicting the model. A less smooth implementation could be a neural network where one defines the network architecture oneself. The latter possibly could yield a performance boost, while taking a long time to implement.

The main outcome of the MR2 artifact is to create common grounds on performance and expectations of the final model.

MR3: Modularity and reusability assessment. Modularity is a debated topic in the ML context, as initially brought up in Section 2.2.3 and later discussed in

Section 4.1.2.

This qualitative artifact aims to evaluate which modules in the code stack from previous projects might be feasible to reuse in the new use case and which modules can be created during development, to decrease the amount of redundant work throughout the project and for future projects. This implies that the artifact is mapped to software complexity. This artifact originates from the partner company that has built a repository structure regarding data collection, data preprocessing, etc. which is being reused for most models that they build (more context in Section 4.1.2).

> The main outcome of the MR3 artifact is a map of which modules from other projects can be reused together with the creation of an overview of the high-level logic of the code, methods, and files that need to be developed with the reused modules taken into account given the use case at hand.

4.2.2 Data collection, data cleaning & data labeling

In this section, an artifact to assess the ML development phases Data Collection, Data Cleaning, and Data Labeling visualized in Figure 2.1 is presented. As the partner company, and most likely other companies too, do not strictly follow these phases for every model that they develop, we decided to couple them together as the metrics and assessments are relevant for all of the data phases presented by Amershi et al. [18] in Figure 2.1.

D1: Data assessment. As data plays a central role in the development of ML systems, data assessment is seen as software complexity since it relates to how people interact with a piece of software in an ML context. This also relates to mitigating/ discovering possible data dependencies which are described in Section 2.3.4.

Data Assessment is a qualitative artifact aimed to assess the following sub-artifacts, inspired by Amershi et al. [18] and Chai et al. [31], as presented in Section 2.3.3:

1. Data Discovery
2. Data Cleaning
3. Data Labeling (if applicable)

The first sub-artifact, **data discovery**, aims to facilitate a discovery of the data in terms of availability, quantity, structure, and readiness. The goal is to reach an understanding of what is the status quo in regards to data that will be required to solve the problem at hand.

In many cases, it is likely that practitioners, similar to the partner company in this thesis, store their data in a data warehouse or equivalent. This implies that the ML engineers will have to communicate with the Data Management team in their discovery.

Quantity refers to the quantity of data available to solve the problem at hand. Some models require more data than others and at the same time, one might have to keep the data volume, described in Sections 2.4.1 and 4.1.3, in mind when choosing an algorithm. The worst-case scenario is that the team realizes that you might not be

able to build a sufficient ML model due to the lack of data, but we think that it is better to realize that sooner rather than later.

Structure in this context refers to how the data is stored, meaning in which tables the data can be located and how the tables relate to one another. Does the ML team need to restructure the data in order to use it and how much time would be required to do so are two questions that this sub-artifact aims to cover.

Finally, data readiness refers to how the ML team can access the data they need for the model versus how much they need to involve the data management team in order to do so. An example of what can increase the complexity is the number of sources an ML engineer need to query the data from, as more sources will require more time needed to maintain the queries throughout the development process, which was highlighted by the ML team at the partner company during the unstructured interview.

Data cleaning refers to the quality of the data. This is an assessment of in what shape the data tables are in, which more concretely refers to which cleaning activities such as removing outliers, missing values, etc. need to be conducted in order to have the data ready to train a model. Activities in this sub-artifact include checking the distribution of the data (mean/max/min, etc.).

Finally, the ML team might need to make **data labeling** efforts to make the data useful in some applications. For example, if one wants to build an image recognition model you will need to have plenty of labeled images to train the model. Labeling in this context then refers to the activity of labeling unlabeled pictures in order to train and validate such a machine. This is not applicable for all use cases, but yet important if required.

> The outcome of the activities of this artifact is ultimately to understand and prepare the data to be ready for training. Furthermore, in order to address the software complexity aspect, which mainly relates to data dependencies, we suggest documenting the activities taken in order to make it easier to debug and test the software at a later stage.

4.2.3 Feature engineering

Feature Engineering in the ML development process in Figure 2.1 refers to finding useful features to use in the model. This is an iterative process that takes place in parallel with model training and evaluation.

In this process step, we present two artifacts; Feature analysis (4.2.3) and Data dependencies (4.2.3), both highly connected to one another. The feature analysis can be considered as an initial feature engineering step, where more obvious non-contributing/skewed features can be detected, and a sensibility check of the model is made. The data dependency analysis can then be considered as a deep dive into how the model works in terms of feature interaction.

FE1: Feature analysis is placed in the software complexity category since the artifact is linked to the interaction between a piece of software and the people as

features are the connection between the context and the model type. We see this artifact as an assessment where one conducts a series of analyses. Some of which are more quantitative which creates an indirect coupling to resource cost, which creates a foundation for the higher-level analysis.

This higher-level combined analysis of all sub-activities is then what is referred to as the artifact as a whole. In this artifact, the idea is to check the quality and need of features in order to obtain an optimal set of features. This is where one looks at the distribution and ranges for different features, and handles possible skews and `NaN`-values. For further feature importance analysis, the explanation techniques mentioned in Section 2.3.5, such as the model-agnostic SHAP and PDPs, should be implemented to get clarity on feature contribution and model workings. For a more model-specific analysis, inherently interpretable models' own attributes should be investigated.

FE2: Data dependency analysis is yet another framework artifact that is connected to software complexity since it is connected to how people interact with a piece of software mainly as it is a combined assessment consisting of a number of smaller analyses depending on the context. This artifact is also connected to the former Feature Analysis in Section 4.2.3 as an important activity in this artifact is to try and remove and mitigate underutilized data dependencies connected to features, as mentioned in Section 2.3.4 and displayed in Table 2.1.

For all four cases in the table, as Sculley et al. [35] mentions, a leave-one-out evaluation could be considered in the case where the number of features is not overwhelmingly many and the time for re-running training is manageable. We believe that there is no universal way of telling when there is a reasonable number of features and runtime is manageable, hence this should be evaluated by the ML team on a case-by-case basis.

To avoid bundled features and ϵ-features, the authors of this thesis suggest that the ML team carefully evaluate how distinct the different features are from one another, e.g., if there are a lot of features evaluating the mean of different intervals for input, one should consider if it adds any value to include all such features. Additionally, looking at the plots of the explanation techniques (Section 4.2.3) and inherent methods could provide insight into whether the contribution of a feature is large enough.

As for correlated features, it is usually difficult to detect them directly as causal features out of a correlated pair of features. However, using domain knowledge and intuition one might be able to mitigate the more obvious cases of correlation and thus be able to remove a less impactful feature that correlates with a more impactful one. Since this could cause leaving out a directly causal feature in favor of a non-causal, correlated feature and thus hurt generalization and performance over time, one should be careful with this step. If the time and competence are at hand, and for less obvious cases, we recommend that one instead looks at handling causal inference as mentioned in Section 2.3.4.

This section unveils two artifacts connected to software complexity to tackle complexity connected to features. The outcome is to uncover unintended complexity by investigating the feature interaction of the model.

4.2.4 Training

As training is an iterative process coupled with feature engineering and evaluation, they probably need to be done in parallel. Regardless, we suggest separating the three steps in this framework for the sake of clarity.

T1: Data volume refers to how much data is needed to generate predictions with an accuracy that satisfy the Performance needs artifact, described in Section 4.2.1.

This artifact is placed in the resource cost category as it is connected to execution time and the fact that the artifact can be assessed in absolute terms, apples to apples, with other model types.

We see two potential cases relating to data volume when one develops a model, either you have plenty of data or too little data. Independent of which, we see the need to evaluate how greedy the chosen algorithms are in terms of how much data they consume to come up with stable and accurate predictions. It is certainly easier to deal with the delicate problem of having plenty of data than the opposite, but regardless knowing how much data different models require can be the deal-breaker when assessing and choosing models. Moreover, depending on the model, it can also be worth evaluating when the model performance reaches a plateau, as this implies that more data will not lead to better performance as mentioned in Section 2.4.1 by Chai et al. [46]. This analysis can then serve as means to understand how much data will have to be fed to the model when it has been deployed, to avoid spending hours in creating/collecting, and quality checking data that is not needed in the end.

The outcome of this artifact is to try out different amounts of data and map out how the performance changes. An additional bonus with less data required is that it could have a positive impact on training runtime. If one instead struggles with having too little data it is more instrumental to understand what is the best possible performance given the data at hand for various models and to see if this is sufficient enough for deploying a model.

In summary, this artifact connected to resource cost intends to test the robustness of the models by facilitating experimentation with various amounts of data. The high-level purpose is to decrease the training time of the models.

4.2.5 Evaluation

In the evaluation step of the ML development process, the model is being assessed using the pre-defined metrics from the model requirements and performance needs assessment of the framework.

E1: Training and prediction runtime analysis refers to the analysis of how long it takes to train and generate predictions from the model.

This artifact is placed in the resource cost category as we choose to link it to the execution part of the resource cost definition (see Chapter 1), given the ML context of this thesis.

The aim of this artifact is to be able to compare an absolute time for training and prediction between different models on an apple to apple basis to fuel a trade-off discussion between other artifacts in the framework, such as performance needs and interpretability and explainability.

In addition to what is presented in Section 2.4.2 as the rationale behind utilizing training and prediction time in the framework, one might also experience a loss in efficiency in the ML team if a model takes too long to train. This input came from the unstructured interview with the ML team at the partner company and as time equals cost, especially for scarce resources, this adds to the rationale to why one wants to develop models that do not take too long to train.

In terms of how one can actually measure training and prediction time, there are packages such as `time.time()` in Python that can be used to output the absolute runtime.

> The outcome of this artifact connected to resource cost is to conduct training- and runtime experimentation to compare the performance of the models.

4.2.6 Deployment

Deployment is the second to last step of the ML development process (Figure 2.1) in which the model is being prepared for going into production. Both of the artifacts below are related to software complexity and understandability with different target audiences in mind, where Section 4.2.6 refers to understandability within the ML team and Section 4.2.6 refers to understandability towards external stakeholders.

DE1: Understandability, as defined in Section 2.3.1, is placed in the software complexity category as it is related to how people interact with a piece of software. The artifact is also highly qualitative to its nature as understandability is individual.

As presented as part of the interview analysis in Section 4.1.4 there are currently no formalized model documentation processes adopted by the ML team at the partner company. Given that as much as 70% of the time spent by developers is associated with understanding code (see Section 2.3.1), the process of explaining code should be embedded in the development process also for ML models. By the time this artifact will be executed, the ML team should have a rather large codebase and thus it should be assessed which parts of the codebase should be further explained. The target audience should be other ML developers/generally code-experienced users and hence the documentation can be in the format of extensive readme files, code comments, following coding standards, and Markdown files. The objective should be to increase the understandability of the code and also to review the structure of the code by taking a step back and reviewing the code objectively before moving further

into the deployment phase. It is assumed that the creation of understandability is something that is considered from the first development steps, and thus these artifacts mainly aim to review and improve the understandability by the mentioned activities.

In addition, the outcome of this step may lead to setting up refactorization efforts to maintain the code base and to foster practices such as peer-review before pushing code to master branches.

DE2: Deployment stakeholder understandability is categorized as software complexity as it is connected to how people interact with a piece of software. Similar to Section 4.2.6 above, this artifact is also connected to understandability, but the target audience is different.

As presented in Section 4.1.4 the ML team has no standardized process for documenting the models towards external, often less technically oriented stakeholders, which is what this artifact aims to facilitate. External stakeholders can be the leadership of associated departments, commercial teams, data teams, etc. The objective is to increase the usability of the ML models in production by preparing documentation on the features used, the accuracy, the purpose, and the limitations of the model. The goal is for relevant stakeholders to understand the model on a high level so that they can explain it to third parties in a sufficient manner.

Similarly, it is highly important to understand what the ML model is meant to inform the user about and how the predictions should be interpreted. The outcome of this artifact can be a one-pager addressing the questions previously mentioned in this paragraph together with a meeting with relevant stakeholders before releasing the model to production or in conjunction with it.

> The outcome of the two artifacts in this section connected to software complexity is to decrease the complexity by increasing the understandability of both technical and non-technical stakeholders by adopting means of explanation on both higher- and lower levels.

4.2.7 Model monitoring

In the final phase of the ML development process, the model is continuously maintained to ensure accurate performance of the model over time. This phase of the process was not covered in this thesis as we experienced a lack of theory in the field, in addition to the fact that the partner company did not have that much structure in place covering this phase.

It is naturally important to make sure that the model in production continuously produces high-quality and accurate predictions for the business, which is why we still kept this phase as part of the framework but without presenting any specific artifacts.

4.3 Validation interview analysis

This section presents the relevant outcomes from the validation interviews held with the ML team at the partner company as well as representatives from academia. The section aims to address RQ(**1**), by validating the suggested framework presented in Section 4.2 above.

All six interviewees saw an overarching value in the framework, given the purpose and research questions of our thesis. One key takeaway of the interviews was that it was deemed more important to keep a few artifacts with substantial research and reasoning for them for this early stage of development of the framework, rather than adding many with a low-level explanation. The mapping of the artifacts to the ML development process was satisfactory according to many respondents, especially so according to the ML team that could clearly see how they could apply the framework throughout their working process. Generally, it was difficult to get any clear answers on the applicability of the framework, as all interviewees mentioned this as something that could only be validated through actually applying the framework to a real case. This concern is addressed in the next section of this thesis, namely the experiment in Section 4.4.

4.4 ML model experiment connected to RQ(2)

This section will walk through the results relating to RQ(**2**), when applying the framework to an existing problem at the partner company and experimenting with the current model and two new, candidate models. Since the experimentation takes a stance in the framework, the structure of this section will be pretty much identical to the previous section.

4.4.1 Experiment context

To answer RQ(**2**), the framework was applied to one of the partner company's existing models for solving a specific problem, as well as to two alternative candidate models. The problem to be solved is of a time series type, where the ML-team at the partner company wants to predict how many newspapers will be sold in a day for different stores in order to optimize how many are delivered to each store, using historic sales data. The current modeling technique applied to the problem is XGBoost [55], with the rationale described by the ML team as it being high performing without demanding long training time, handling missing values inherently, and being able to inherently weight features based on feature importance. The resulting predictions are delivered on a weekly basis, which can be tweaked manually before delivery. More information on the three different modeling techniques can be found in Appendix B.1.

4.4.2 Machine learning requirements

This section of the experiment examines existing needs for the model implemented, in order to guide the creation of the two comparison models and also to validate the current XGBoost implementation in terms of interpretability and performance needs.

MR1: Interpretability needs. As this experiment is based on an existing model, the need for interpretability is already evaluated by the ML team. They state that there is a somewhat high need for interpretability, in order to understand and evaluate feature importance, much as they are of the belief that the features could use being re-evaluated. Also, given that manual corrections are often made, we believe that it is important to keep interpretability high to minimize the number of corrections needed by easier troubleshooting in accordance with what Rudin mentions in Section 2.3.5. Therefore, the additional modeling techniques applied in this experiment were a basic linear regression (LR) and a linear generalized additive model (linear GAM) which is mentioned in Section 2.3.5 as inherently interpretable. The linear regression model is highly interpretable as it includes few parameters and models simple, linear relationships between a feature and the target variable. The linear GAM is interpretable as it is an extension of general linear models, that allows for non-linear feature functions while keeping the high level of interpretability.

MR2: Performance needs. As the ML team and the output receivers at the partner company are satisfied with the performance of the existing XGBoost model implementation, we decided that the XGBoost model's performance serves as a good baseline for performance for both tweaks of the original implementation as well as the other tested ML techniques. The existing model uses mean absolute error (MAE) as an evaluation metric, which will be the metric used to compare performance for the experiment rounds.

Given that the print prediction problem is rather simple, that the current XGBoost model is described to function rather well, and finally due to the scope being limited, we, in discussions with the ML team at the partner company, decided that rather simple models using library imports rather than large code implementations with computationally heavy lines of code and long training runtimes were desirable. This also maps nicely to the model choices based on the interpretability needs, as there are multiple Python libraries for fully functioning models at hand for both implementations.

MR3: Modularity assessment. Given that the ML team has a number of models in production, they have already implemented solid modularity (see Section 4.1.2), which is further used for tweaks to the current model as well as the two additional modeling techniques.

The modularity implemented by the ML team was useful when creating the new models, as the setup for querying data, and calculating features were completely reused, while portions of the method for training were reused. Due to basing the experiment on a fully developed, modular GitHub repository, no additional modularity was added during this artifact check.

The outcome of this step showed that there is a wish for simpler, more interpretable models for the problem at hand performing at least on par with the existing XGBoost implementation. Finally, the existing modularity in the ML team repository was deemed sufficient for further use.

4.4.3 Data Collection, Data Cleaning & Data Labeling

This section includes a summary of the data assessment steps conducted at the partner company for the existing model.

D1: Data assessment. As this experiment is based on an existing model, the data assessment has already been carried out by the ML team in collaboration with relevant stakeholders. As for data readiness, explained as a portion of the data discovery phase in Section 4.2.2, the ML team used to retrieve the data on a daily basis in form of a file that was read into the model.

During our time at the partner company they, together with the data team, changed the single data source to having Big Query as an interface for retrieving the data from a live updated Google Cloud Platform. The rationale for this change was that the new storage allowed faster data retrieval and feature calculation, as there was no longer a need to read a static file, and some of the data filterings could be done through querying.

As for the data cleaning step, the data retrieved by the ML team is to a large extent cleaned. There can be lags in reports of the number of magazines sold per store, which the code accounts for by removing stores with missing sold numbers for the last two weeks or more.

The last step of the data assessment artifact, data labeling, is not applicable to this particular problem, as the data source is historic sales data with the number of sold magazines reported in the data.

The outcome of this step was mainly a change of data source to retrieve faster data collection and feature calculation, as other steps were already conducted by the ML team.

4.4.4 Feature engineering

This section accounts for our analysis of the features pre-defined by the ML team at the partner company, together with a data dependencies evaluation. As mentioned in section 4.2.3, this step is conducted iteratively and in parallel with the training and evaluation step.

FE1: Feature analysis. As this experiment is based on an existing model, the initial feature analysis, as well as a sanity check of the features, has already been conducted by the ML team.

Our assumption when first getting familiar with the model and its features was that there were perhaps an excessive amount of mean and median features for different

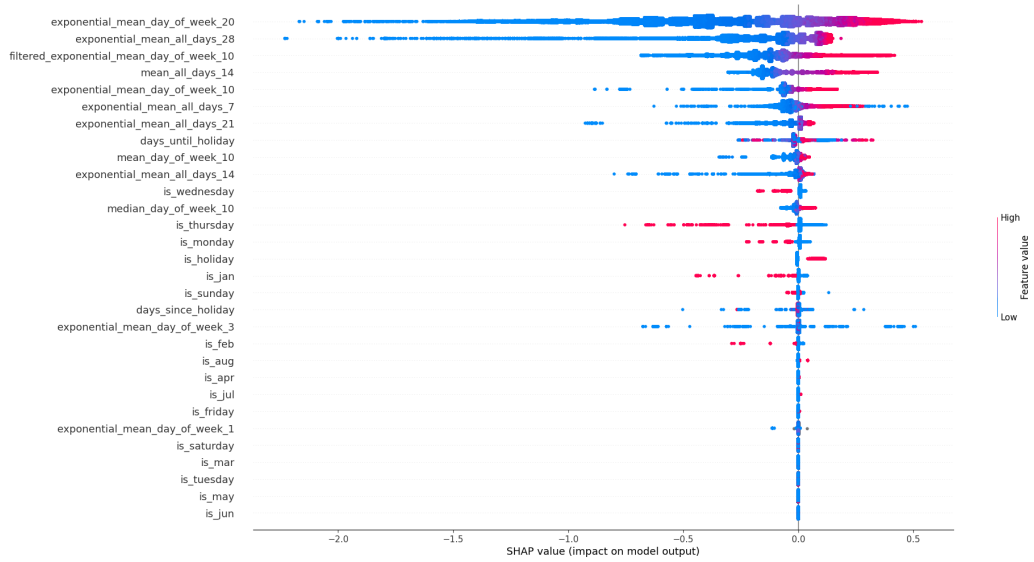

Figure 4.3: XGBoost SHAP plot, where features are ranked based on importance, blue colors indicate low feature values, and red high ones. The x-axis corresponds to the impact on model output (SHAP-value).

intervals, given the simplicity of the problem. This, together with the implementation of two alternative models, made this phase relevant for exploring, even with the initial feature work by the ML team. Since there can be missing sales data in the input, as mentioned in Section 4.4.3, there exist missing values for some features as well.

The existing XGBoost model learns branch directions for missing values during training, while the other two models do not have inherent methods for handling missing values. Instead, iterative imputer was used, a Scikit Learn package for handling missing feature values by estimating them as a function of other features [62].

As mentioned in Section 4.2.3, explanation techniques such as SHAP and PDP plots can be used for feature analysis. An example of a SHAP plot can be seen in Figure 4.3. The interpretation of it is that the features are ranked, with the top showing the most important feature according to the SHAP values, and each point for a feature corresponds to a sample's feature value. A blue value equals a low feature value and red a high one, and its placement indicates if its SHAP value, i.e., impact on the model output, is low or high (from left to right). The partial dependence plots, where an example can be found in Figure 4.4, show for an increasing feature value on the x-axis, the corresponding partial dependence for the model output on the y-axis.

XGBoost feature analysis. For the XGBoost model, both the inherent feature importance plot in Figure 4.5 as well as the model agnostic SHAP plot in Figure 4.3 indicate that the different mean and median features (of historic sales data) are the most impactful to the predictions.

Indifference to the feature importance plot, the SHAP plot shows some interesting patterns, e.g., the features *is_holiday* and *is_thursday*. The SHAP plot indicates

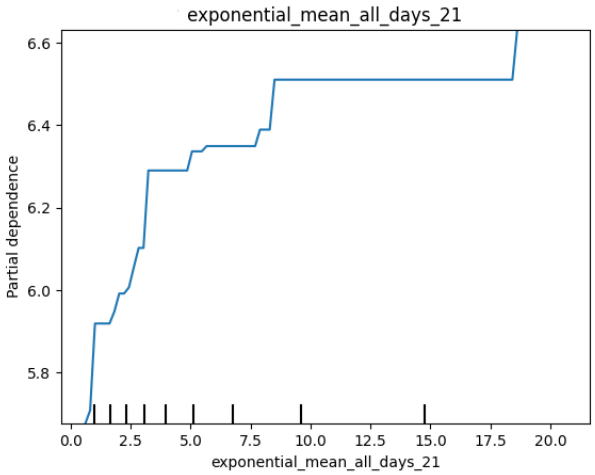

Figure 4.4: PDP for the feature *exponential_mean_all_days_21* of the XGBoost model, displaying partial dependence of the model output for different values of the feature.

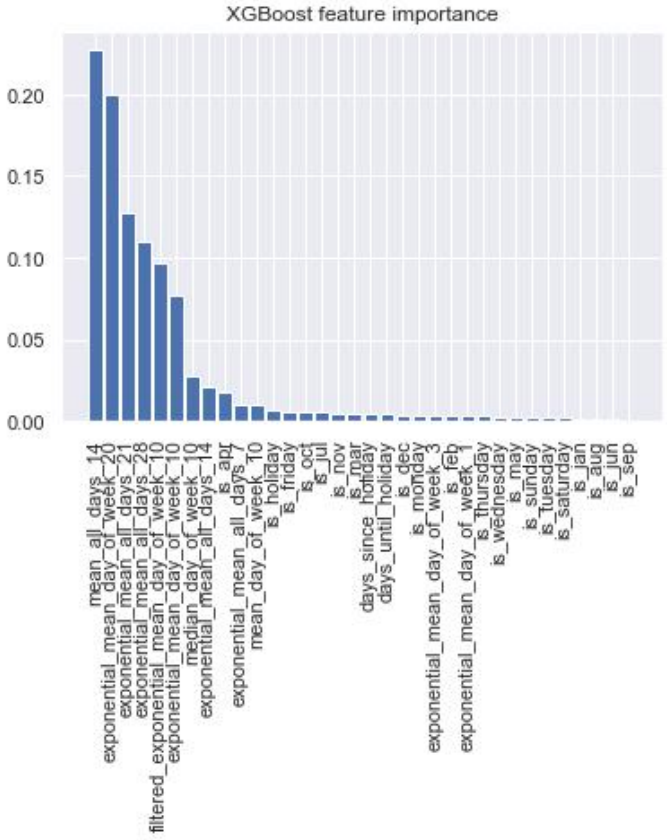

Figure 4.5: XGBoost feature importance in declining order.

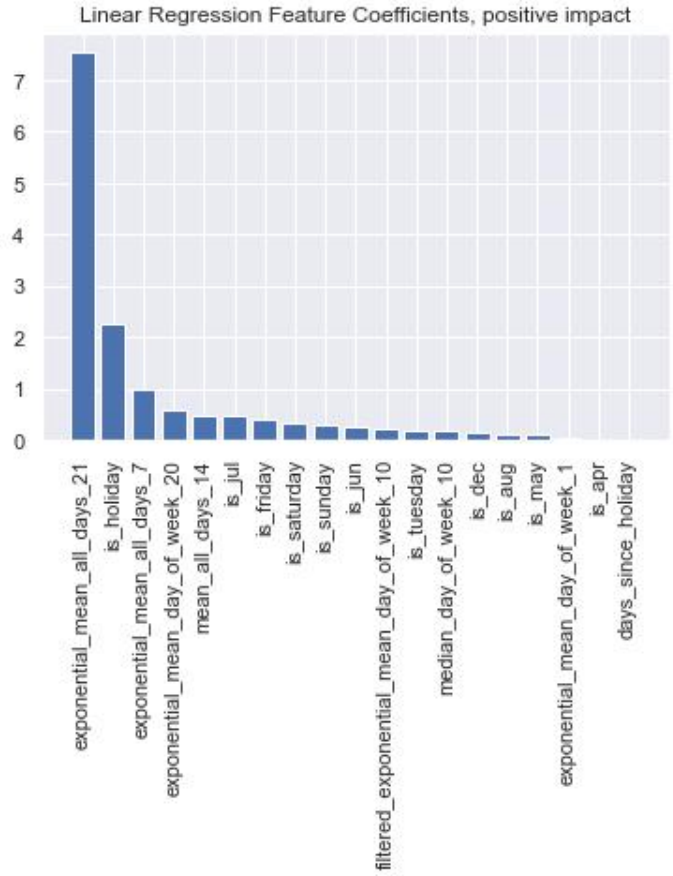

Figure 4.6: LR coefficients with a positive impact on the predicted value, from most to least positive impact.

that sales go up during holidays and that if it is a Thursday, the sales are probably lower than on an average day.

Looking more closely at one of the more impactful features according to both Figure 4.5 and Figure 4.3, *exponential_mean_all_days_21*, we see the PDP of this feature in Figure 4.4. This graph confirms that the higher the mean, the higher the prediction. In general, all the PDP plots for the XGBoost model made sense intuitively and when compared to the suggested importance by the SHAP and feature importance plots. A number of the binary features that were lower ranked were flat in their corresponding PDP, suggesting no to little impact on the predicted outcome.

Linear Regression feature analysis. For the LR model, the results of the plots of model coefficients in Figure 4.6 and Figure 4.7 are less intuitive.

The bars in Figure 4.6 show features that for a higher feature value, the prediction gets higher, and the other way around in Figure 4.7.

While supposedly the feature with the most positive impact is a mean feature, we see in Figure 4.7 that the top-2 most negatively impacting features are also two mean features. The SHAP plot in Figure 4.8 matches the coefficient plots with the top features, and also indicates that most features have a negligible impact on the predicted output. However, since we have multiple mean features for different but

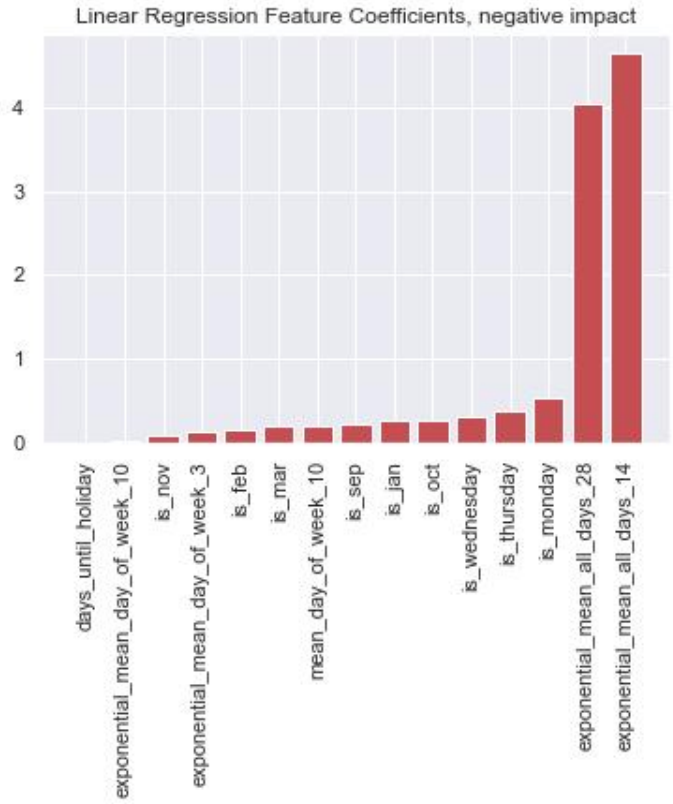

Figure 4.7: LR coefficients with a negative impact on the predicted value, from least to most negative impact.

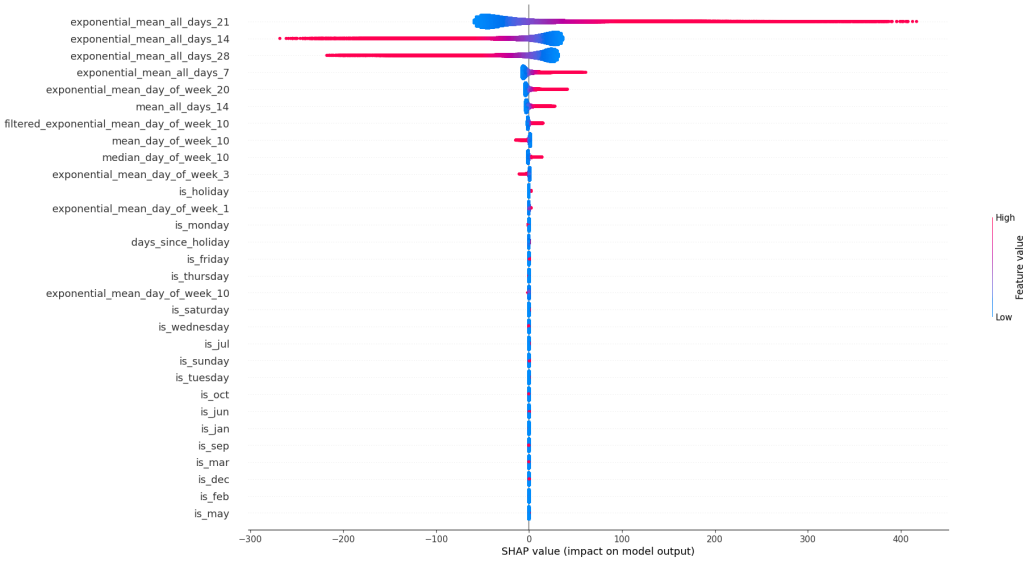

Figure 4.8: Linear Regression SHAP plot, where features are ranked based on importance, blue colors indicate low feature values, and red high ones. The x-axis corresponds to the impact on model output (SHAP-value).

```
LinearGAM
=============================================== ==========================================================
Distribution:                      NormalDist Effective DoF:                                     315.3648
Link Function:                  IdentityLink Log Likelihood:                                -1316736.3316
Number of Samples:                    400907 AIC:                                            2634105.3927
                                             AICc:                                           2634105.894
                                             GCV:                                                 10.1511
                                             Scale:                                               10.1368
                                             Pseudo R-Squared:                                     0.8832
=============================================================================================================
Feature Function                  Lambda               Rank         EDoF         P > x        Sig. Code
================================= ==================== ============ ============ ============ ============
s(0)                              [0.001]              25           22.9         1.11e-16     ***
s(1)                              [0.001]              25           19.1         1.11e-16     ***
s(2)                              [0.001]              25           19.6         1.11e-16     ***
s(3)                              [0.001]              25           19.6         1.11e-16     ***
s(4)                              [0.001]              25           23.7         1.11e-16     ***
s(5)                              [0.001]              25           23.7         1.11e-16     ***
s(6)                              [0.001]              25           23.6         1.11e-16     ***
s(7)                              [0.001]              25           23.0         1.11e-16     ***
s(8)                              [0.001]              25           1.0          1.11e-16     ***
```

Figure 4.9: Snapshot of the Linear GAM model summary, including statistics for the model (top part) as well as for each feature function (bottom part). The full model summary can be found in Appendix C.1.

overlapping intervals our data is multicollinear. Multicollinearity might not affect the performance of an LR model, but the reliability in the individual effects of each feature (i.e., the coefficients displayed in Figure 4.6 and Figure 4.7) might be lost [63]. Therefore, we are careful to draw any conclusions on what is displayed in the plots.

Linear GAM feature analysis. The `pygam` implementation of the linear GAM provides a model summary, displaying information about each feature function as well as the model as a whole. The column *Rank* in Figure 4.9 shows the degree of non-linearity for each feature function before smoothing is applied (by a factor of column *Lambda*), while *EDoF* shows the final degree of non-linearity. The model summary does not display the feature names, but by comparing the training data with named columns, we can see that *EDoF* is high for all the mean features (ranging between 19 and 24, with a high level of non-linearity), while the binary features have an *EDoF* of 1, indicating a linear relationship, which makes sense given that a binary feature only can take on two different values.

For the linear GAM model, the PDPs are an inherent function in the library that is used. The PDP displaying the same feature as in Figure 4.4, *exponential_mean_all_days_21*, for the XGBoost-model can be seen in Figure 4.10. Confirmed by the model summary, we see a highly non-linear relationship between the feature and the prediction. The feature curve is not that wiggly, and the confidence interval is pretty narrow, indicating a solid PDP. In general, all the PDP plots for the linear GAM were stable and had a pretty narrow confidence interval.

Some of the mean features, however, as with the linear regression model, display a declining trend—the higher the mean, the lower the predicted outcome. This could be due to concurvity, an extension of the multicollinearity problem described for the

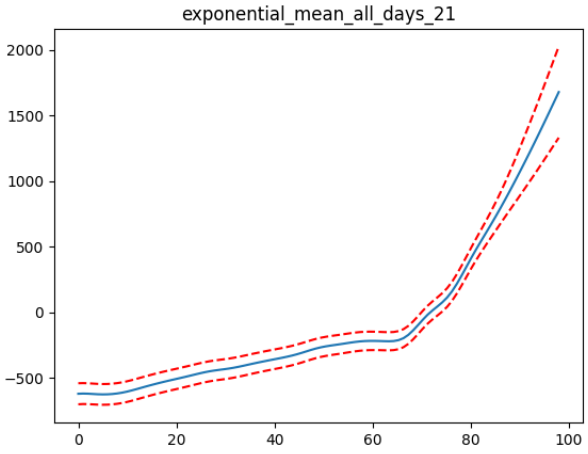

Figure 4.10: PDP with 95% confidence interval for the feature *exponential_mean_all_days_21* of the linear GAM, displaying partial dependence of the model output for different values of the feature.

LR model.[1]

To summarize, the XGBoost model is considered the most reliable and sensible in terms of feature analysis and feature importance. The linear GAM also provides some sensible insights from the PDP plots and shows robust results, apart from some features. The model summary of the linear GAM adds additional insight into the level of non-linearity for each feature function, with high degrees of non-linearity for most continuous features, suggesting that an LR model with only linear relationships might not be optimal for the task at hand. The LR model on the other hand gives less interpretable feature results, resulting in a difficulty to analyze feature importance and impact.

FE2: Data dependencies. The ML team at the partner company mentioned that the current features of the XGBoost model are created on intuition, which makes us believe that there is possibly some legacy, bundled and/or ϵ-features (as described in Table 2.1) in the data. Given that we have multiple features for mean calculations, our dataset is likely to have a high correlation. But since we probably do not have the issue where for a pair of correlated features, i.e., a pair of mean features, only one is directly causal, we will not investigate data dependencies in the form of correlated features further.

The leave-one-out analysis was restricted to a subset of the mean features of the models. This was due to it being too time-consuming to conduct a leave-one-out analysis of all (34) features, and the selection was limited to mean features as these require computation and thus, likely more runtime to calculate, unlike the binary

[1]Concurvity occurs when a smoothing term of a feature could be approximated by another feature smoothing term, which is likely due to the same reasoning as in the linear regression case [64]. Again, it does not affect the performance but can hurt the interpretation.

features.

Although we did not conduct a full leave-one-out analysis, we believe that for the sake of testing the framework and suggesting further work for the ML team, this was value-adding. We decided to conduct a leave-one-out analysis for two of the lowest-ranked mean features, *exponential_mean_day_of_week_1*, and *exponential_mean_day_of_week_3*, from the analysis in Section 4.4.4 of Figure 4.5 and Figure 4.3 for the XGBoost model.

These runs resulted in unchanged performance but a decrease in both feature generation time and training time of ∼20% each. Given the LR model/linear GAM analyses in Section 4.4.4 indicating possible multicollinearity/concurvity, making the interpretation of the feature analysis less reliable, we decided to conduct the leave-one-out analysis for the same features for these two models as indicated by the XGBoost plots.

Doing the same for the LR model also resulted in unchanged performance and ∼20% decrease in feature generation time. However, the training runtime remained approximately the same as when including all features. Lastly, for the linear GAM, we saw the same results as for the XGBoost—unchanged performance and ∼20% decrease in feature generation time and training runtime.

For all three models, removing both features at the same time yielded the same results as the individual exclusion runs. These leave-one-out analyses indicate that these features add little to no improvement to any of the three implemented models, while they add resource cost in form of increased training runtime for the XGBoost model and linear GAM. Hence in the case of the LR model, we can consider these two features as bundled, i.e., a group adding little/negligible value.

For the other two models, they might even be considered as ϵ-features as they increase the training runtime and thus the resource cost. Given the results of this leave-one-out analysis, we recommend the ML team at the partner company to keep conducting the same analysis for other features of less importance according to Section 4.4.4.

The main outcome of this step is that there are some features that should be removed, yielding a decrease in both feature generation and training runtime, without affecting the performance.

4.4.5 Training

This section covers experimenting with data volume for the existing XGBoost model and the two other implemented models, with a comparison between the changes in performance for different data volumes for each of the three models.

T1: Data volume. Given that the problem at hand is a time series problem and given the fact that there is a lot of data available, we decided to experiment with data volume by decreasing the interval of dates used for training.

As can be seen in Table 4.1, the different models do not handle reduced training

Table 4.1: Performance for the three different models for different sizes of the data used for training.

	Data Volume (size in megabytes)	% of original data volume	Performance on validation set (MAE w. std error)	Absolute difference MAE compared to original data size	Absolute difference MAE compared to XGBoost w. original data size
XGBoost	163.9	100%	1.71 (0.0032)	-	-
	123.0	75%	1.72 (0.0031)	0.01	0.01
	81.9	50%	1.73 (0.0032)	0.02	0.02
LR	163.9	100%	1.76 (0.0032)	-	0.05
	123.0	75%	1.77 (0.0032)	0.02	0.06
	81.9	50%	1.82 (0.0033)	0.06	0.11
GAM	163.9	100%	1.77 (0.0033)	-	0.06
	123.0	75%	2.69 (0.0073)	0.92	0.98
	81.9	50%	11.54 (0.0450)	9.77	9.83

data equally well. The XGBoost model performs roughly the same training on one year of sales data (100% in the table) as when training on just six months of data, a reduction of the training dataset by 40.9 MB.

The XGBoost model with 50% of the data also outperforms the other two model implementations regardless of the amount of data used, even if so just marginally in many cases. The LR model gets a slightly bigger decrease in performance for each reduction step, while the linear GAM performs poorly on 50% of the data, and also noticeably worse for 75% of the data.

The fact that the linear GAM performance is notably worse when dataset size decreases are probably explained by the fact that it tends to overfit when the training dataset size is too small. The feature functions tend to be too wiggly, resulting in poorer generalization than when a large enough dataset is used. As the LR model only models simple linear relationships between the features and output variable, it is less prone to overfitting, which could explain the consistency in performance (together with possibly small seasonality in the sales numbers).

The main takeaway from this step is that the XGBoost model remains robust for different sizes of the training data used, while the LR model performs slightly worse for each reduction and lastly the linear GAM is highly sensitive to reductions.

4.4.6 Evaluation

In this step, apart from looking at the performance (measured in MAE for this experiment) of the models, the training and prediction runtimes of the different models are analyzed and compared.

E1: Training and prediction runtime analysis. For this artifact, we logged performance, runtime for training for all three models on the original size data set, excluding any explanation techniques, and lastly the prediction time for these models. As can be seen in Table 4.2, the prediction runtime is small, especially in comparison to training time, and on par for all three models. This is likely due to the small interval used for prediction—only one week ahead of today's date is

Table 4.2: Performance and runtime in seconds for the three different models on the original dataset size.

Model	Performance (MAE w. std error)	Runtime Training	Runtime Training Relative to XGBoost	Runtime Prediction	Runtime Prediction Relative to XGBoost
XGBoost	1.71 (0.0032)	270	-	0.10	-
LR	1.76 (0.0032)	17	0.01x	0.11	1.09x
Linear GAM	1.77 (0.0033)	2199	8.14x	0.15	1.48x

predicted and sent out to the team responsible for sending out magazines to stores. Reasonably, this will remain unchanged as the predictions are time-sensitive, and it would make little sense to send out predictions for larger ranges.

Additionally, as mentioned in Section 4.2.5, the ML team at the partner company considers long training runtimes to be a bottleneck in their day-to-day work. As can be seen in Table 4.2, there is a vast difference in training runtime for the three different models. The LR model takes one-hundredth of the time the XGBoost takes, while the linear GAM is eight times slower than XGBoost. The fact that training the LR model takes much less time than the other two was no surprise, as it models every feature as having a simple linear relationship to the output.

The slow training runtime for the linear GAM could possibly be explained by that the model is allowed to specify the function parameters of each feature's relationship to the output by a grid search during training. The XGBoost model also shrinks training runtime by not evaluating the regularization parameters used at each node, but instead growing the tree to max depth and then pruning before the regularization parameters are evaluated. As can be seen in Table 4.2, the XGBoost model performs better, looking at the chosen evaluation metric, than the other two models.

> The main takeaway from this step is the relative training runtimes of the three different models, where the LR model takes by far the least amount of runtime for training and the linear GAM is significantly slower than the other two models. The prediction runtime was negligible and on par for all three models.

4.4.7 Deployment

Since further work with these models and interaction with the stakeholders that receive predictions from the model is handled by the ML team at the partner company, the artifacts in this section will account for practices taken in order to give a smooth handover to the ML team for further work.

DE1: Understandability. As this artifact is supposed to increase the understandability of ML engineers and other code-experienced users, towards the end we held a session with the ML team where we walked through the code function by function, explaining all steps taken.

In case of any unclarities, we either modified the code or added additional comments in the Python files. To make the code as clear as possible before this session, we followed the PEP-8 style guide for all code written, used logical naming for

variables and functions, and also commented on functions where the code might be less intuitive. Given that a large amount of the repository was built by the whole ML team at the partner company, it is worth noting that a rather high level of understandability of the repository was already obtained before our work.

DE2: Deployment stakeholder understandability. As the output of the models consists of a file of predictions sent out to the relevant stakeholder (the team who decides on the deliveries and thus also the number of magazines to print), rather than deploying the actual model as an API, this step might not be as critical as in other cases. However, the ML team is recommended to create readme files for models that have multiple stakeholders as receivers and where the models are read through APIs.

> The main outcome of this step was the session held to hand over the implementation made during the thesis, to give a higher level of understandability for the ML team as they take over and further develop the models.

4.4.8 Experiment results summary

To summarize the results of the work related to RQ(**2**), three different models were analyzed; an existing XGBoost model as well as two alternative models developed by us, a linear regression (LR) model and a linear generalized additive model (linear GAM), all three with a rather high level of inherent interpretability which was one of the main rationales for choosing the alternative models.

The existing XGBoost model performance in mean absolute error (MAE) was used as a performance comparison, and all three models were evaluated for different sizes of the dataset used for training. Using the original dataset size and all the original features, the performance in terms of MAE for all features was on par for all three models, which can also be seen in the residual plot in Figure 4.11.

The models were also compared in terms of training and prediction runtime as well as interpretability and explainability results for model features. The XGBoost model consistently performed better than the other two models in terms of MAE and also showed greater robustness to decrease in the dataset size. In terms of interpretability, the XGBoost model results were reasonable, and also the linear GAM results were reasonable for a majority of the features. The LR model showed some less intuitive feature impacts, likely due to a high level of multicollinearity of the training data. The LR model however had a significantly shorter training runtime than the other two models.

> The partner company is recommended to keep the existing XGBoost model, although considering using less data for training and also removing some features, and conducting further feature and data dependencies analysis.

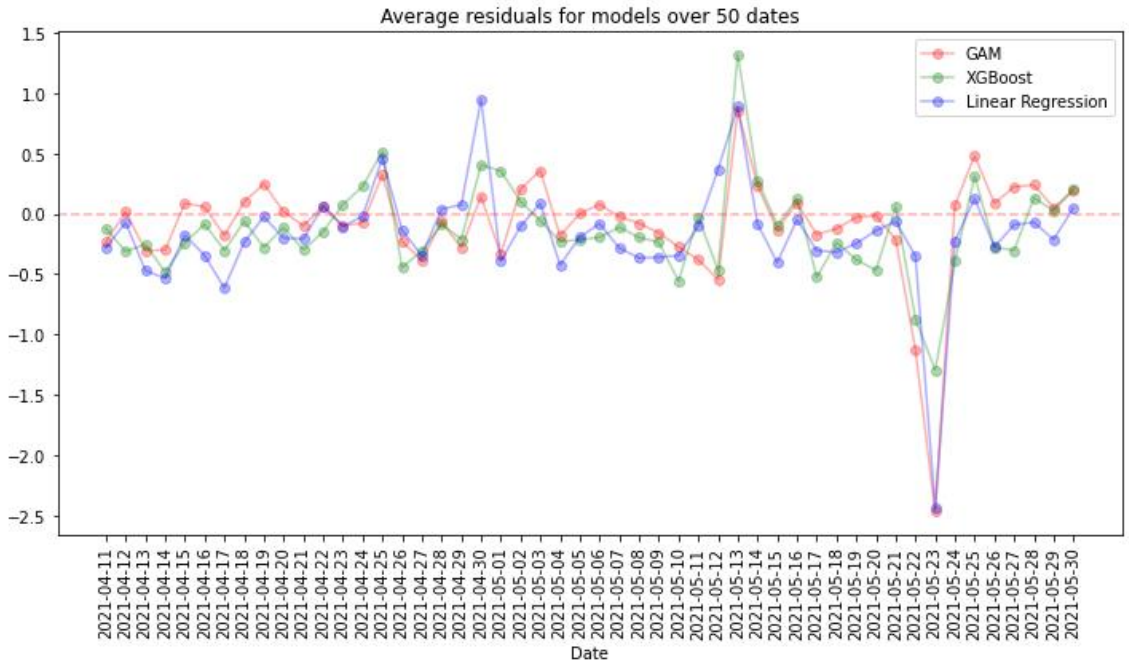

Figure 4.11: Average residuals (difference between the true value and predicted value) for the three models over 50 dates.

5

Discussions & Limitations

The objective of this thesis was to develop a framework for tackling ML model complexity, which also formed the two research questions. The framework consists of a number of artifacts originating from both the theory as well as the unstructured and semi-structured interviews with members of the ML team at the partner company, Bonnier News, and independent researchers, respectively. As the framework was developed the most important concerns have been a) do all artifacts make sense according to various sources, b) are there any artifacts missing to make the framework generalizable and complete, and c) how useful is the framework to contexts outside of the partner company.

The first two concerns, i.e., a) and b), we addressed in the validation interviews where we asked open questions to the ML team and the independent researchers. Obviously, time was restricted and the parties spent different amounts of time with the pre-read version of the report, shared with them in advance, which made the outcome of the interviews limited. We were satisfied with the outcomes of this method step in the scope of our work but there are certain limitations connected to a) and b) that we did not catch. Regarding c) this is certainly difficult for us to assess. Again, the validation interviews were meant to address parts of this concern, but we did never interview representatives from other companies to test the framework in other contexts. The semi-structured framework validation interviews were the closest we came to assessing the framework with external resources and generally they did see the value in it. Some of them did say that it was difficult for them to assess the usability, but they did find it valuable to gather artifacts the way we did it in the framework. To summarize, it will require future research to prove or reject the usability of the framework in other contexts outside of the partner company.

In both of our research questions, we used the expression *trade-off* between resource cost and software complexity. In RQ(**1**), the idea was to identify artifacts that optimize ML models concerning this trade-off and in RQ(**2**) the objective was to assess how successful the artifacts are in prioritizing this trade-off, applied to an ML model.

Given the definition of *trade-off*s presented in Chapter 1, we would like to reiterate the issue of trade-offs. Given that the existing trade-offs will vary depending on the use case, the framework does not state what trade-offs exist given a specific artifact. We believe that including such explicit statements would possibly add confusion, and hence chose to exclude them. The key contribution to the trade-off discussion is the

optimization and categorization of different artifacts, which was validated through both validation interviews and the conducted experiment. To further contextualize, we will now present a discussion on possible trade-offs given the experiment phase of this thesis. One trade-off could be between the level of interpretability and training runtime. In our model experiment in Section 4.4 we implemented a linear GAM, mentioned by many as a highly interpretable model, contrasting with the slightly less interpretable tree model XGBoost. However, the runtime for the linear GAM was significantly larger than for the XGBoost model (see Table 4.2), indicating a higher level of interpretability at the cost of longer training runtime. However, it is unclear if this trade-off holds for the general case as, e.g., neural network models usually take a large training time while also being less interpretable than most other machine learning models. We, during our experiment, also found that the level of interpretability of the linear GAM vs. the XGBoost model was similar; the model summary of the linear GAM added insight into the explained degree of non-linearity for a feature function, but on the other hand, the feature importance plot of the XGBoost model added even more straightforward insight on model workings in our opinion.

Another trade-off worth mentioning is the possible one between performance needs and data volume. In the experiment, we had the delicate problem of having large amounts of training data at hand. For especially the linear GAM, but also to an extent the LR model, we saw that decreasing the dataset size hurt the model performance. In the general case, one usually has more issues with collecting enough data, but the trade-off most likely remains as more data often equals a better-performing model. However, sometimes this trade-off might be less relevant—as was the case with the XGBoost model implementation, where the performance remained robust for significant decreases in dataset size.

Additionally to the trade-off between resource cost and software complexity described in Chapter 1, we also discovered some trade-offs within software complexity. A trade-off within the software complexity category could be between great performance and a high degree of interpretability. This does not mean that interpretable models cannot yield good performance, but more so that choosing, e.g., a neural network could yield even better performance. However, as neural networks are not inherently interpretable, the choice comes down to prioritizing between the trade-off of either high interpretability or optimal performance. If we have a complicated problem where interpretability is not the main requirement and we care about optimal performance, we might choose to disregard interpretability in favor of great performance and go with a neural network. On the other hand, for a simpler problem such as the one experimented with in Section 4.4, the performance of simpler ML algorithms is good enough and thus the trade-off perhaps is not as present as in other cases. This is also the reason why we place this assessment early in the development process, simply because we see the value of discussing this with the primary stakeholders as early as possible in the project to mitigate the risk of developing the wrong type of model due to not being aware of important requirements. In the experiment phase, it is also interesting to see that our choice to develop an LR model due to its high level of interpretability was not successful in terms of

interpretability. This is as the dataset suffers from multicollinearity, yielding results of the interpretability check that are not reliable.

Another trade-off could be between performance needs and data assessment. It is difficult to develop a model with great performance without having an appropriate amount of high-quality data that is accessible to the team. The trade-off is how good performance one can reach with the available data to not oversell the capabilities of the model before actually implementing it. Similarly, knowing about this can help to steer the development project to mitigate risks associated with bad data quality for example.

An important note to highlight in our opinion is that the trade-offs found in this thesis are highly dependent on a) how we choose to define software complexity and resource cost, b) how we choose to map artifacts into either software complexity or resource cost, and finally, c) what artifacts are included and which are not included in the final framework. For a), even slightly different definitions would have possibly changed which of the two different definitions the artifacts mapped best to. For b), there could be a discussion regarding if some artifacts solely map to one of the two definitions. It could be that an artifact to some extent maps to both, or perhaps should be mapped to the other definition than it is now. Finally, for c), from our validation interviews we could not find any clearly missing artifacts, making us believe that this factor should not impact the trade-off discussion.

For RQ(**2**), the objective was to assess the framework in an environment as close to the real world as possible. This was done by applying it to an existing model, provided by the partner company and compared against two other models developed by us on the same problem and use case. The scope of the thesis did not allow us to develop models from scratch, which would have been interesting as that would have given us the opportunity to go through all artifacts end-to-end. That would potentially have given us the chance to challenge all artifacts to a greater extent, which obviously would have strengthened the usefulness of the framework.

Also related to RQ(**2**) we found it a bit difficult to follow the framework during the development phases feature engineering, training, and evaluation as these three steps are highly iterative in its nature. That means that the artifacts need to be conducted iteratively, a mechanism somehow difficult to visualize in an appropriate way. We experienced this ourselves in the experiment which indicates one valuable item coming out of that method step.

Another concrete outcome of the experiment, more so towards the partner company, was the results of the feature analysis and data volume artifacts where we could show features that could be considered redundant given the plots that were outputted and that the XGBoost model performed equally well using just 50% of the training data. The value of these two discoveries is that it would take less time to train and run the model, given that the redundant features are removed and that the ML team use less training data. These two concrete items indicate the relevance of the framework, as they decrease both software complexity (feature engineering artifacts findings) and resource cost (data volume artifact findings). They show how implementing our framework, even on existing models in production, contributes concrete value to the

partner company.

In general terms, we were satisfied with how the research was conducted in the course of the thesis as we believe that we were able to find a balance between different methods to answer both of the research questions. The first set of discovery interviews, which are referred to as unstructured interviews, were conducted early on in the process to understand the context of the relevant teams at the partner company. They served the purpose and helped us to build an initial understanding fast but at the same time, we could have structured them more diligently as that would have allowed us to present the outcome of them in a more structured way. Given the unstructured setup, we had to ask plenty of follow-up questions as we went on writing the report which may have been avoided. Moreover, we did not record the sessions but instead relied on our notes alone. This is something we learned from, and when we later conducted the semi-structured interviews we did record them (after consent from the interviewees) to support the notes.

5.1 Limitations

In this section, the limitations associated with research conducted in the course of this thesis are presented. The section is structured into limitations related to the two research questions, respectively.

5.1.1 Limitations related to RQ(1)

A limitation connected to the first research question is that the monitoring phase of the ML development process (see Figure 2.1) is considered to be out of scope for this thesis. That is because a) the time constraint of the project did not allow for us to spend the time required to output well-thought and proven outcomes, b) that we found out when conducting initial research that this part of the process seemed bigger than expected, and not as researched as other parts of the process, and finally, c) the partner company model provided to us was not in live deployment.

A second limitation is connected to the definition of resource cost, *the measure of the resources expended by another system in interacting with a piece of software, where the interacting system is another machine. Resource cost is defined by the execution time and storage needed to conduct the computation.* The 'execution time' we translated into the training and prediction runtime artifact but we do not have similar interpretations of 'storage' in any of the artifacts. To include 'storage' as part of the framework we experimented with memory consumption and GPU/CPU. We included memory consumption as an artifact until we, in the scope of the experiment, decided to exclude it as we did not manage to interpret how the outcome influenced resource cost. Regarding GPU/CPU, it was mentioned during the framework validation interviews that it might be interesting to look at. After an analysis, we decided to leave it out due to time constraints.

5.1.2 Limitations related to RQ(2)

A limitation related to RQ(**2**) is that we were not able to fully test all artifacts presented in the framework in the experiment phase. This was due to the fact that the framework is meant to be used throughout the full development process, while the experiment conducted in this thesis started from an already existing model provided by the ML team at the partner company. This means that we were not able to fully validate whether the artifacts MR1, MR2, MR3 (ML requirements phase), and D1 (data collection, cleaning, and labeling phase) at the beginning of the process and DE2 (deployment phase) at the end of the process made complete sense in practical terms. We bridged this gap primarily with the results from the validation interviews which cover RQ(**1**), but not RQ(**2**). This is connected to the limitation that we did not build completely new models given a specific problem to validate the framework using it the way it is intended to be used. The reason for this is simply that it did not fit into the scope of the project and hence we had to tackle the experiment by utilizing existing models in an existing repository to speed up the development process.

Another limitation related to RQ(**2**) is that the problem that formed the experiment method of the thesis provided by the partner company was comparably simple. One can argue that this was beneficial as we did not have to spend a lot of time understanding the problem, but on the other hand, it made the experiment slightly trivial in some cases. For example, in the feature engineering phase, we did not tackle causality as the nature of the features (mainly the mean features) did not allow for further investigation. Similarly, unlike many other ML problems, there was plenty of data available for the given problem. In addition, the data was already cleaned and made available to us in an appropriate way. Both of these properties, the amount of data, and the fact that the data was already cleaned can be seen as limitations as they made the experiment around the artifacts tackling these two potential complexities difficult to assess thoroughly. On the other hand, the theory, the unstructured, and semi-structured interviews all served as means to help us bridge this potential lack of validation to some extent.

6

Threats to Validity

Given that this thesis is conducted in a field study setting it implies that we face low generalizability of findings [51]. This suggests that the framework, which is the main outcome of this thesis, might be either a) incomplete or b) that it is not useful to other practitioners than the partner company with which it was developed, also known as a threat to external validity [65]. While external validity in the paper by Feldt and Magazinius [65] is mentioned in the context of quantitative research and we in this paper also conduct qualitative research, we still do see this as a potential threat to validity. The former we deal with by conducting validation interviews with independent researchers who were not involved in the thesis project. We presented the framework and asked open-ended questions to catch missing pieces, uncertainties, or other concerns related to the framework. The latter is a bit more difficult to address as the scope and time constraints of the project did not allow us to go out and conduct studies with other practitioners. On the other hand, the collaboration with the partner company did allow us to capture realism in the framework which for us means to make it as useful as possible for practitioners. The capturing of realistic context is also mentioned by Stol and Fitzgerald [51] as one of the main advantages of field studies.

Krishna et al. [66] present a list of "bad smells" in software engineering research (SE), i.e., indicators of deeper problems, their impact on the reliability of a study as well as how to mitigate them. Not all bad smells reported in [66] are considered relevant for this thesis, but the subset that is will be addressed here. The first relevant bad smell mentioned is "being boring", with the impact that the research conducted has negligible SE impact. The mitigation strategy mentioned for this is the dialogue with practitioners, which our validation interviews with practitioners as well as representatives from academia account for. The second bad smell mentioned is not using related works, with the impact of possible duplication, using outdated benchmarks, and finally not using state-of-the-art methods. This too was mitigated by starting the research of this thesis with a thorough literature search. Another bad smell mentioned is inadequate reporting, with the impact of the study not being reproducible. This threat of reliability maps to our experiment to address RQ(**2**), as we do not provide the source code for the different models implemented. However, the aim of this part of the thesis is not to reproduce the exact experiment conducted, but more so to contextualize how the framework can be applied in a real-world setting. Having an underpowered experiment is also mentioned as a bad smell, leading to an overestimation of the effect size and possibly replication problems. In

the context of this thesis, we see the issue of only trying out the framework on one problem at a single company as a possible underpowered experiment. It is not given that the effects of the framework displayed in the conducted experiment indicate a general effect of applying the framework. This could be mitigated by trying out the framework for other problems and for more companies.

7

Conclusions & Future research

The objective of the thesis was to contribute to the field of SE4ML by researching SE and ML resource cost and software complexity, and based on this research provide a framework for ML engineers to use as guidance throughout the development process to tackle model complexity. The two research questions that were formed to serve this purpose were:

1. What are suitable artifacts for optimizing ML models concerning resource cost/software complexity trade-off?
2. Applied to an ML model, how successful are the artifacts in prioritizing the resource cost/software complexity trade-off?

7.1 Conclusion of RQ(1)

Artifacts that are suitable for optimizing ML models concerning the resource cost and software complexity trade-off can be found in Figure 4.1 and a more tangible description of each artifact, the input, the output, and the purpose can be found in Figure 4.2. The optimization and prioritization of each artifact (see Section 4.2) will affect how later artifacts are prioritized, which in the end will steer how models are developed with respect to complexity. For example, the performance needs artifact will affect later artifacts, e.g., feature analysis. The trade-off part of the question is related to how resource cost and the rest of the artifacts related to the software complexity category are balanced. This is something that the framework helps display and guide by the structure it provides for evaluation of the different artifacts, as the existing trade-offs are case-specific. In short, an example of such trade-offs can be between interpretability needs (see Section 4.2.1) and training and prediction runtime analysis (see Section 4.2.5), where the more inherently interpretable model (the GAM) had a much longer training and prediction runtime compared to XGBoost, indicating that such a trade-off does exist.

For some artifacts on the other hand, such as understandability in software complexity, no clear trade-off to any of the resource cost artifacts was found. Interestingly, we also found trade-offs between artifacts in the same software complexity category, for example between performance needs (Section 4.2.1 and interpretability needs (Section 4.2.1). Hence, the framework does surface trade-offs both between resource cost and software complexity artifacts as well as between artifacts in the software complexity category which equip the practitioners with the means to optimize rel-

evant trade-offs given their specific use case, fulfilling the objective of the research question. However, there are most likely trade-offs that we did not surface in the course of the research that may or may not be context-specific. Additionally, trade-offs can be more or less significant depending on model choice and context.

7.2 Conclusion of RQ(2)

When we applied the framework to an existing model and two other models on the same problem we did find the framework to be useful, especially in terms of the structure it creates for tackling complexity-related concerns connected to model development. In terms of the prioritization of resource cost and software complexity, there is no distinct objective measure developed by us to assess this. Instead, as iterated on in Section 4.4 the main outcome is the mapping of requirements, need-to-have vs. nice-to-have in the initial part of the process and using this as guidance throughout the development process.

We did find that the framework was useful when applying it specifically for feature analysis and on the data volume experiments. For some parts of the ML development process, from feature engineering through evaluation, it was a bit more difficult to follow the framework in a structured way given the iterative nature of these phases. Worth noticing, mentioned in limitations (Section 5.1.2), given the context of the experiment we were not able to apply the framework end-to-end which may have uncovered other weaknesses or strengths.

Ultimately, we do find the framework successful for what it is purposed to do. We do certainly see the limitations with it, for example, the generalizability to other contexts but we do see it as an appropriate first version, which was also indicated by the interviewees in the framework validation interviews.

7.3 Future research

As indicated in the discussion (Chapter 5), the main concerns that we partly addressed in the scope of the thesis, but that yet need to be further researched, can be summarized into; a) whether the artifacts presented as part of the framework are complete or whether more need to be added and, b) what is the applicability of the framework for other ML teams and at other company contexts to further generalize the framework. We would suggest performing a larger study including a number of different tech companies to gain insights into whether the framework and the artifacts make sense in more contexts. Such a study would tackle both of the concerns by validating the usability of the framework in other contexts as well as investigating whether there are artifacts missing or if any of the existing artifacts need to be tweaked to improve generalizability.

Another interesting future research area is connected to the monitoring phase of the ML development process, which was excluded from the scope of this thesis. First, as mentioned in Section 5.1.1, it would certainly be interesting to expand the framework to also include this phase and to address which artifacts should be

included in the monitoring phase to further tackle complexity. Second, it would be interesting to understand how monitoring, being the last phase of the process, is affected by applying the framework throughout the development process.

Lastly, as mentioned in Limitations (Section 5.1.1), 'storage' is excluded from the thesis. Although it was not an essential problem in the context of the partner company in this thesis, our assumption is that it is possibly a key element at many other companies and hence it could provide valuable insight looking further into this resource cost in future work.

Bibliography

[1] Fumihiro Kumeno. "Sofware engneering challenges for machine learning applications: A literature review". In: *Intelligent Decision Technologies* 13 (Feb. 2020), pp. 463–476. DOI: 10.3233/IDT-190160.

[2] Kristian Kersting et al. "SE4ML-Software Engineering for AI-ML-based Systems (Dagstuhl Seminar 20091)". In: *Dagstuhl Reports*. Vol. 10. 2. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2020.

[3] Joseph P. Kearney et al. "Software Complexity Measurement". In: *Commun. ACM* 29.11 (Nov. 1986), pp. 1044–1050. ISSN: 0001-0782. DOI: 10.1145/7538.7540. URL: https://doi.org/10.1145/7538.7540.

[4] Victor R Basili. "Qualitative software complexity models: A summary". In: (1980).

[5] *About Bonnier News*. https://www.bonniernews.se/om-oss/. Accessed: 2021-10-14.

[6] *Bonnier News - front page*. https://www.bonniernews.se/. Accessed: 2021-10-14.

[7] *Framework definition and meaning: Collins english dictionary*. URL: https://www.collinsdictionary.com/dictionary/english/framework.

[8] *Suitable definition according to Cambridge Dictionary*. URL: https://dictionary.cambridge.org/dictionary/english/suitable.

[9] *Optimize definition according to Cambridge Dictionary*. URL: https://dictionary.cambridge.org/dictionary/english/optimize.

[10] *Prioritize definition according to Cambridge Dictionary*. URL: https://dictionary.cambridge.org/dictionary/english/prioritize.

[11] *Trade-off definition according to Cambridge Dictionary*. URL: https://dictionary.cambridge.org/dictionary/english/trade-off.

[12] Cynthia Rudin. "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead". In: *Nature Machine Intelligence* 1.5 (2019), pp. 206–215.

[13] Lucy Ellen Lwakatare et al. "A Taxonomy of Software Engineering Challenges for Machine Learning Systems: An Empirical Investigation". In: Apr. 2019, pp. 227–243. ISBN: 978-3-030-19033-0. DOI: 10.1007/978-3-030-19034-7_14.

[14] M. I. Jordan and T. M. Mitchell. “Machine learning: Trends, perspectives, and prospects”. In: *Science* 349.6245 (2015), pp. 255–260. DOI: 10.1126/science.aaa8415. URL: https://www.science.org/doi/abs/10.1126/science.aaa8415.

[15] Chen, Chiang, and Storey. “Business Intelligence and Analytics: From Big Data to Big Impact”. In: *MIS Quarterly* 36.4 (2012), p. 1165. DOI: 10.2307/41703503. URL: https://doi.org/10.2307%5C%2F41703503.

[16] Meenu Mary John, Helena Holmström Olsson, and Jan Bosch. “Developing ML/DL Models: A Design Framework”. In: *Proceedings of the International Conference on Software and System Processes*. ICSSP ’20. Seoul, Republic of Korea: Association for Computing Machinery, 2020, pp. 1–10. ISBN: 9781450375122. DOI: 10.1145/3379177.3388892. URL: https://doi.org/10.1145/3379177.3388892.

[17] W. S. Humphrey. “The Software Engineering Process: Definition and Scope”. In: *SIGSOFT Softw. Eng. Notes* 14.4 (Apr. 1988), pp. 82–83. ISSN: 0163-5948. DOI: 10.1145/75111.75122. URL: https://doi.org/10.1145/75111.75122.

[18] Saleema Amershi et al. “Software engineering for machine learning: A case study”. In: *International Conference on Software Engineering (ICSE 2019) - Software Engineering in Practice track*. May 2019. URL: https://www.microsoft.com/en-us/research/uploads/prod/2019/03/amershi-icse-2019_Software_Engineering_for_Machine_Learning.pdf.

[19] Charles Hill et al. “Trials and tribulations of developers of intelligent systems: A field study”. In: *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 2016, pp. 162–170. DOI: 10.1109/VLHCC.2016.7739680.

[20] Rob Ashmore, Radu Calinescu, and Colin Paterson. “Assuring the Machine Learning Lifecycle: Desiderata, Methods, and Challenges”. In: *CoRR* abs/1905.04223 (2019). arXiv: 1905.04223. URL: http://arxiv.org/abs/1905.04223.

[21] Zhiyuan Wan et al. “How does machine learning change software development practices?” In: *IEEE Transactions on Software Engineering* (2019).

[22] Rajiv D. Banker, Srikant M. Datar, and Dani Zweig. “Software complexity and maintainability”. In: *Proceedings of the Tenth International Conference on Information Systems*. 1989, pp. 247–255.

[23] Gerardo Canfora and Aniello Cimitile. “Software Maintenance”. In: *Handbook of Software Engineering and Knowledge Engineering* 1 (Jan. 2001). DOI: 10.1142/9789812389718_0005.

[24] Edward E. Ogheneovo. “On the Relationship between Software Complexity and Maintenance Costs”. In: vol. Vol.02No.14. 2014, p. 16. DOI: 10.4236/jcc.2014.214001. URL: //www.scirp.org/journal/paperinformation.aspx?paperid=51631.

[25] Asher Trockman et al. “"Automatically Assessing Code Understandability" Reanalyzed: Combined Metrics Matter”. In: *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. 2018, pp. 314–318.

[26] Karamjit Kaur et al. “Static and dynamic complexity analysis of software metrics”. In: *World Academy of Science, Engineering and Technology* 56 (Aug. 2009), pp. 159–161.

[27] Barry W Boehm, John R Brown, and Mlity Lipow. “Quantitative evaluation of software quality”. In: *Proceedings of the 2nd international conference on Software engineering*. 1976, pp. 592–605.

[28] Xia Hu et al. “Model complexity of deep learning: a survey”. In: *Knowledge and Information Systems* 63 (Oct. 2021), pp. 1–35. DOI: 10.1007/s10115-021-01605-0.

[29] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. “Quantifying Model Complexity via Functional Decomposition for Better Post-hoc Interpretability”. In: *Communications in Computer and Information Science* (2020), pp. 193–204. ISSN: 1865-0937. DOI: 10.1007/978-3-030-43823-4_17. URL: http://dx.doi.org/10.1007/978-3-030-43823-4_17.

[30] D. Sculley et al. “Machine Learning: The High Interest Credit Card of Technical Debt”. In: *SE4ML: Software Engineering for Machine Learning (NIPS 2014 Workshop)*. 2014.

[31] Chengliang Chai et al. “Data Management for Machine Learning: A Survey”. In: *IEEE Transactions on Knowledge and Data Engineering* (2022), pp. 1–1. DOI: 10.1109/TKDE.2022.3148237.

[32] Neoklis Polyzotis et al. “Data Lifecycle Challenges in Production Machine Learning: A Survey”. In: *SIGMOD Rec.* 47.2 (Dec. 2018), pp. 17–28. ISSN: 0163-5808. DOI: 10.1145/3299887.3299891. URL: https://doi.org/10.1145/3299887.3299891.

[33] Sebastian Schelter et al. “On challenges in machine learning model management”. In: *IEEE Data Engineering Bulletin* 41 (2018), pp. 5–15.

[34] Yiming Tang et al. “An empirical study of refactorings and technical debt in machine learning systems”. In: IEEE, May 2021. DOI: 10.1109/icse43902.2021.00033.

[35] David Sculley et al. “Hidden technical debt in machine learning systems”. In: *In Advances in neural information processing systems* (2015), pp. 2503–2511.

[36] Peter Spirtes. *Introduction to Causal Inference*. Oct. 2010. URL: https://www.jmlr.org/papers/volume11/spirtes10a/spirtes10a.pdf.

[37] Leilani H Gilpin et al. “Explaining explanations: An overview of interpretability of machine learning”. In: *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*. IEEE. 2018, pp. 80–89.

[38] Toshiki Mori and Naoshi Uchihira. “Balancing the trade-off between accuracy and interpretability in software defect prediction”. In: *Empirical Software Engineering* 24.2 (2019), pp. 779–825.

[39] Ričards Marcinkevičs and Julia E Vogt. “Interpretability and explainability: A machine learning zoo mini-tour”. In: *arXiv preprint arXiv:2012.01805* (2020).

[40] Meike Nauta et al. “From Anecdotal Evidence to Quantitative Evaluation Methods: A Systematic Review on Evaluating Explainable AI”. In: *arXiv preprint arXiv:2201.08164* (2022).

[41] Cynthia Rudin et al. “Interpretable machine learning: Fundamental principles and 10 grand challenges”. In: *Statistics Surveys* 16 (2022), pp. 1–85.

[42] Zachary C Lipton. “The Mythos of Model Interpretability: In machine learning, the concept of interpretability is both important and slippery.” In: *Queue* 16.3 (2018), pp. 31–57.

[43] Avi Rosenfeld and Ariella Richardson. “Explainability in human–agent systems”. In: *Autonomous Agents and Multi-Agent Systems* 33.6 (2019), pp. 673–705.

[44] Alex Goldstein et al. “Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation”. In: *journal of Computational and Graphical Statistics* 24.1 (2015), pp. 44–65.

[45] *4.1. partial dependence and individual conditional expectation plots*. URL: https://scikit-learn.org/stable/modules/partial_dependence.html.

[46] Michael Chai et al. *Notes from the AI frontier insights from hundreds of use cases*. Apr. 2018. URL: https://www.mckinsey.com/~/media/mckinsey/featured%5C%20insights/artificial%5C%20intelligence/notes%5C%20from%5C%20the%5C%20ai%5C%20frontier%5C%20applications%5C%20and%5C%20value%5C%20of%5C%20deep%5C%20learning/notes-from-the-ai-frontier-insights-from-hundreds-of-use-cases-discussion-paper.ashx.

[47] Ekaba Bisong. “Google BigQuery”. In: *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. Berkeley, CA: Apress, 2019, pp. 485–517. ISBN: 978-1-4842-4470-8. DOI: 10.1007/978-1-4842-4470-8_38. URL: https://doi.org/10.1007/978-1-4842-4470-8_38.

[48] Mark Mucchetti. “Managing BigQuery Costs”. In: *BigQuery for Data Warehousing: Managed Data Analysis in the Google Cloud*. Berkeley, CA: Apress, 2020, pp. 61–71. ISBN: 978-1-4842-6186-6. DOI: 10.1007/978-1-4842-6186-6_4. URL: https://doi.org/10.1007/978-1-4842-6186-6_4.

[49] T S Lim, Wei-Yin Loh, and Yu-Shan Shih. “A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-Three Old and New Classification Algorithms”. In: *Machine Learning* 40 (Sept. 2000), pp. 203–228. DOI: 10.1023/A:1007608224229.

[50] Kajaree Das and Rabi Narayan Behera. “A Survey on Machine Learning: Concept,Algorithms and Applications”. In: *International Journal of Innovative Research in Computer and Communication Engineering* 5 (2017).

[51] Klaas-Jan Stol and Brian Fitzgerald. “The ABC of Software Engineering Research”. In: *ACM Transactions on Software Engineering and Methodology* 27.3 (Sept. 2018). ISSN: 1049-331X. DOI: 10.1145/3241743.

[52] Harris M Cooper. "Scientific guidelines for conducting integrative research reviews". In: *Review of educational research* 52.2 (1982), pp. 291–302.

[53] S.E. Hove and B. Anda. "Experiences from conducting semi-structured interviews in empirical software engineering research". In: *11th IEEE International Software Metrics Symposium (METRICS'05)*. 2005, 10 pp.–23. DOI: 10.1109/METRICS.2005.24.

[54] Joanna F DeFranco and Phillip A Laplante. "A content analysis process for qualitative software engineering research". In: *Innovations in Systems and Software Engineering* 13.2 (2017), pp. 129–141.

[55] *XGBoost documentation*. URL: https://xgboost.readthedocs.io/en/stable/index.html.

[56] *Sklearn.linear$_m$odel.linearregression*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html.

[57] URL: https://pygam.readthedocs.io/en/latest/api/gam.html.

[58] *Time - Time Access and conversions*. URL: https://docs.python.org/3/library/time.html.

[59] Laurie Williams et al. "Strengthening the case for pair programming". In: *IEEE software* 17.4 (2000), pp. 19–25.

[60] *Python enhancement proposals*. URL: https://www.python.org/dev/peps/pep-0008/.

[61] *What is mob programming?* Mar. 2021. URL: https://www.agilealliance.org/glossary/mob-programming/#q=~(infinite~true~filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'mob*20programming))~searchTerm~'~sort~false~sortDirection~'asc~page~1).

[62] *6.4. imputation of missing values*. URL: https://scikit-learn.org/stable/modules/impute.html#iterative-imputer.

[63] *Multicollinearity: Detecting multicollinearity with VIF*. Apr. 2020. URL: https://www.analyticsvidhya.com/blog/2020/03/what-is-multicollinearity/#:~:text=Fixing%5C%20Multicollinearity-,What%5C%20is%5C%20Multicollinearity%5C%3F,variable%5C%20in%5C%20a%5C%20regression%5C%20model..

[64] URL: https://stat.ethz.ch/R-manual/R-devel/library/mgcv/html/concurvity.html.

[65] Robert Feldt and Ana Magazinius. "Validity threats in empirical software engineering research-an initial survey." In: *Seke*. 2010, pp. 374–379.

[66] Rahul Krishna et al. "Bad Smells in Software Analytics Papers". In: *Information and Software Technology* 112 (Mar. 2018). DOI: 10.1016/j.infsof.2019.04.005.

[67] *Gradient boosting algorithm: A complete guide for beginners*. Oct. 2021. URL: https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-beginners/.

[68] Tianqi Chen and Carlos Guestrin. *XGBoost: A scalable tree boosting system - arxiv*. URL: https://arxiv.org/pdf/1603.02754.

[69] *Linear Regression*. URL: http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm.

[70] *Multiple Linear Regression*. URL: http://www.stat.yale.edu/Courses/1997-98/101/linmult.htm.

[71] *6.1 - Introduction to Glms: Stat 504*. URL: https://online.stat.psu.edu/stat504/lesson/6/6.1.

[72] Kim Larsen. *Gam: The predictive modeling silver bullet*. July 2015. URL: https://multithreaded.stitchfix.com/assets/files/gam.pdf.

A

Appendix 1

A.1 Questions asked in semi-structured interviews

A.1.1 Warm-up

1. How many years have you been working in the industry?
2. How many years of research experience do you have?
3. How many years of experience do you have in the Software Engineering field?
4. How many years of experience do you have in the Machine Learning field?

A.1.2 Main questions

1. Is the purpose of this framework reasonable? Why / Why not?
2. What is your spontaneous reaction immediately after we presented the framework?
3. Is there any part of the framework that does not make sense? What exactly does not make sense? How can this be fixed?
4. Is there any artifact that is not included but that should be? In which part of the development process? Why?
5. Is there anything else related to the framework that is not included but that should be? (could be artifacts, purpose, clarifications, etc.)

B

Appendix 2

B.1 ML model descriptions

B.1.1 XGBoost

`XGBoost` (eXtreme Gradient Boosting) is an open-source, gradient-boosting algorithm library [55]. The gradient boosting algorithm builds an ensemble of decision trees, i.e. it builds trees sequentially, where each sequential tree tries to rectify the error residuals of the previous one [67]. The library authors [68] describe the XGBoost model as follows; The XGBoost model predicts the outcome for sample i as:

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i) \tag{B.1}$$

Where each f_k corresponds to an independent decision tree, with tree structure q and weights w. Each leaf in a decision tree contains a continuous score, denoted by w_i for leaf i. The final prediction for a sample i is then obtained by summing up the scores of all leaves. The objective to be minimized, i.e. the loss function, in order to learn the set of functions F (the set of decision trees) is as follows:

$$L(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \text{ ,where } \Omega(f) = \gamma T + \frac{1}{2}\lambda ||w||^2 \tag{B.2}$$

The first term in B.2 corresponds to the same differentiable loss function that is used for gradient tree boosting, and measures the difference between the predicted $\hat{y}_i$ and true value y_i. The second term is a regularization parameter specific to XGBoost (with T corresponding to the number of leaves in the tree), which penalizes the complexity of the model. The regularization parameter also helps prevent overfitting, by smoothing the final weights. The model is then trained in an additive fashion, where for iteration t we want to minimize the following objective:

$$L^{(t)} = \sum_{i=1}^{n} l(\hat{y}_i, y_i^{(t-1)}) + f_t(x_i) + \Omega(f_t) \tag{B.3}$$

So we greedly add the tree f_t that gives the greatest improvement according to B.2. For further details on the XGBoost model workings and parameters, please see the library authors' paper [68] or the XGBoost model documentation [55].

B.1.2 Linear Regression

Linear regression is a modeling technique that models the relationship between two variables (the explanatory variable x and the response variable y) [69]. Given that we have multiple explanatory variables, i.e. features, we in this thesis implemented multiple linear regression. Instead of modeling the relationship between two variables, we model the relationship between multiple explanatory variables and the response variable [70]. The equation for multiple linear regression for one sample i is as follows:

$$y_i = \beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip} \tag{B.4}$$

Where y_i is the response variable for sample i, β_0 is the intercept, and β_{ik} is the calculated coefficient for explanatory variable k. The best-fitting line in the used scikit-learn Python package for linear regression [56] is then calculated using ordinary least squares (OLS). OLS minimizes the sum of squares of the deviations between the line and each true datapoint [69].

B.1.3 Linear General Additive Model

Generalized linear models (GLMs) is an extension of general linear models (conventional OLS as described above). Compared to general linear models, GLMs do not assume that the output is normally distributed [71]. Generalized additive models (GAMs) in turn expand GLMs by allowing non-linear feature functions [72]. As GAMs maintain additivity from GLMs, one can easily examine the effect of each feature on the response variable, while holding all other features constant. The general form of a GAM can be written as [57]:

$$\begin{aligned} g(\mu|X) &= \beta_0 + f_1(x_1) + ... + f_n(x_n) \\ y &\sim ExponentialFamily(\mu|X) \end{aligned} \tag{B.5}$$

Given B.5 a GAM has three different components:

1. Distribution from the exponential family
2. Link function g(.)
3. Functional form

The link function g(.) is the function that links the expected value to the linear prediction. The functional form denotes nonparametric, smooth functions, which are the feature functions that model the linear or non-linear impact of a feature on the response variable [57]. These feature functions are built using penalized basis splines, a piecewise polynomial curve. The penalization encourages smoother curve estimates by minimizing the penalized sum of squares. The nonparametric term means that the feature function is fully determined by the data rather than a set of parameters [72]. In this thesis, the common model `LinearGAM` from the `pyGAM` library was implemented. The `LinearGAM` has a normal distribution and the identity link function [57], and the implemented models strength of smoothing penalty was optimized using the inherent `.gridsearch()`-method.

C

Appendix 3

C.1 Linear GAM model summary

LinearGAM

Distribution:	NormalDist	Effective DoF:	315.3648
Link Function:	IdentityLink	Log Likelihood:	-1316736.3316
Number of Samples:	400907	AIC:	2634105.3927
		AICc:	2634105.894
		GCV:	10.1511
		Scale:	10.1368
		Pseudo R-Squared:	0.8832

Feature Function	Lambda	Rank	EDoF	P > x	Sig. Code
s(0)	[0.001]	25	22.9	1.11e-16	***
s(1)	[0.001]	25	19.1	1.11e-16	***
s(2)	[0.001]	25	19.6	1.11e-16	***
s(3)	[0.001]	25	19.6	1.11e-16	***
s(4)	[0.001]	25	23.7	1.11e-16	***
s(5)	[0.001]	25	23.7	1.11e-16	***
s(6)	[0.001]	25	23.6	1.11e-16	***
s(7)	[0.001]	25	23.0	1.11e-16	***
s(8)	[0.001]	25	1.0	1.11e-16	***
s(9)	[0.001]	25	1.0	1.11e-16	***
s(10)	[0.001]	25	1.0	1.11e-16	***
s(11)	[0.001]	25	1.0	1.11e-16	***
s(12)	[0.001]	25	1.0	1.11e-16	***
s(13)	[0.001]	25	1.0	1.11e-16	***
s(14)	[0.001]	25	0.0	1.11e-16	***
s(15)	[0.001]	25	1.0	1.11e-16	***
s(16)	[0.001]	25	1.0	1.11e-16	***
s(17)	[0.001]	25	1.0	1.11e-16	***
s(18)	[0.001]	25	1.0	1.11e-16	***
s(19)	[0.001]	25	1.0	1.11e-16	***
s(20)	[0.001]	25	1.0	1.11e-16	***
s(21)	[0.001]	25	1.0	1.11e-16	***
s(22)	[0.001]	25	1.0	1.11e-16	***
s(23)	[0.001]	25	1.0	1.11e-16	***
s(24)	[0.001]	25	1.0	1.11e-16	***
s(25)	[0.001]	25	1.0	1.11e-16	***
s(26)	[0.001]	25	0.0	1.11e-16	***
s(27)	[0.001]	25	1.0	1.62e-03	**
s(28)	[0.001]	25	17.9	1.11e-16	***
s(29)	[0.001]	25	17.1	1.11e-16	***
s(30)	[0.001]	25	16.1	1.11e-16	***
s(31)	[0.001]	25	22.9	1.11e-16	***
s(32)	[0.001]	25	24.0	1.11e-16	***
s(33)	[0.001]	25	24.0	1.11e-16	***
intercept		1	0.0	1.11e-16	***

Figure C.1: Linear GAM model summary, including statistics for the model (top part) as well as for each feature function (bottom part).