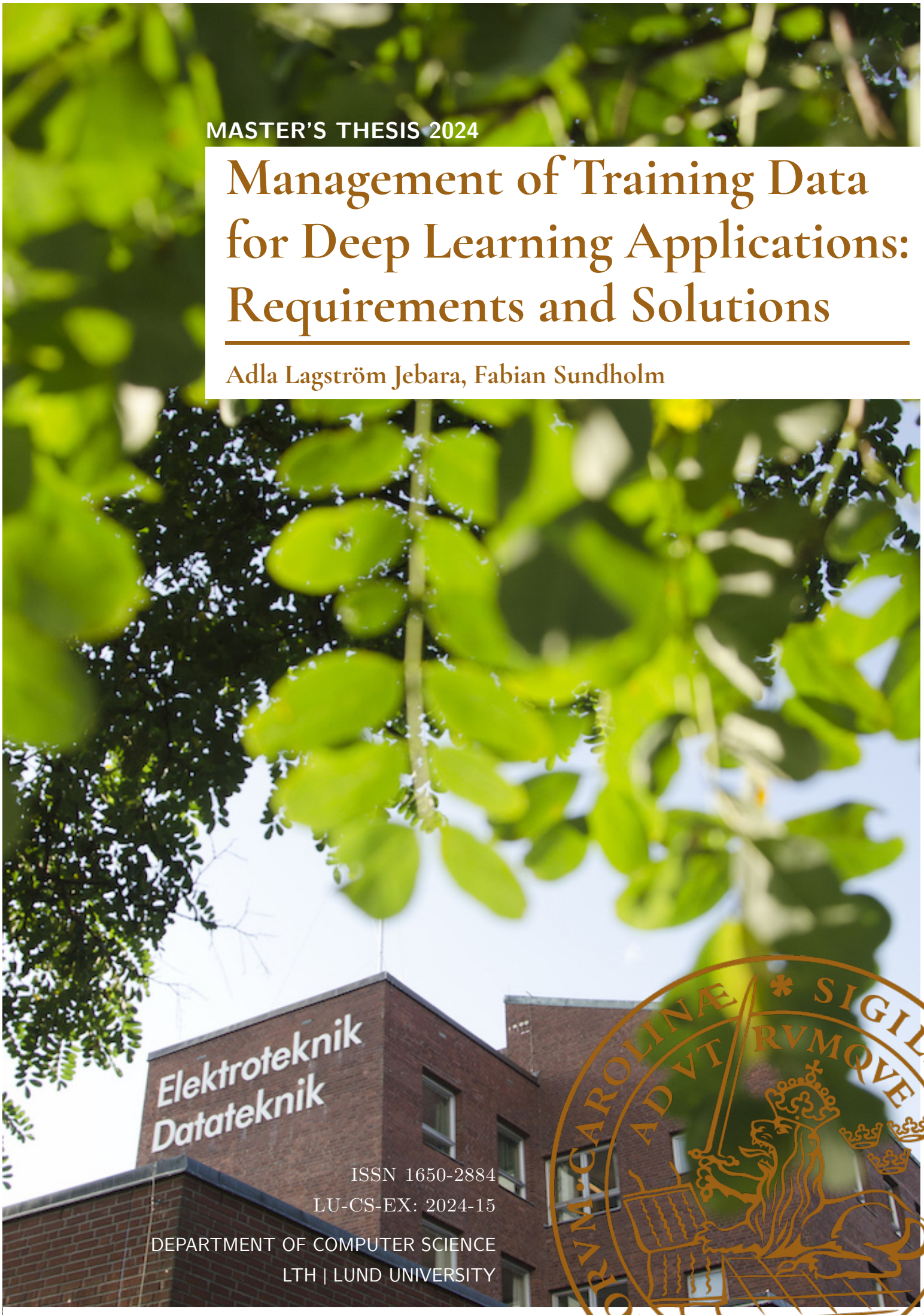


MASTER'S THESIS 2024

Management of Training Data for Deep Learning Applications: Requirements and Solutions

Adla Lagström Jebara, Fabian Sundholm

ISSN 1650-2884
LU-CS-EX: 2024-15

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2024-15

Management of Training Data for Deep Learning Applications: Requirements and Solutions

Adla Lagström Jebara, Fabian Sundholm

Management of Training Data for Deep Learning Applications: Requirements and Solutions

Adla Lagström Jebara
nod14aj1@student.lu.se

Fabian Sundholm
mat14fsu@student.lu.se

April 3, 2024

Master's thesis work carried out at Precise Biometrics.

Supervisors: Lars Bendix, lars.bendix@cs.lth.se
Fredrik Stål, fredrik.stal@precisebiometrics.com

Examiner: Emelie Engström, emelie.engstrom@cs.lth.se

Abstract

In the realm of software development, extensive research has been conducted on source code management, but little to no attention has been given to managing associated data, such as the large volume of training data needed for the development of deep learning applications. This thesis aims to investigate if there is a scalable solution for storing and managing training data used in different variants of machine learning models. This research includes identifying and formulating requirements for a training data management system, proposing design solutions to address these requirements, and finally, implementing a proof of concept. The requirement specification was formulated through literature reviews and developer interviews. Design solutions were developed in alignment with the identified requirements and by exploring available tools. Thereafter, one of the two design solutions was chosen for implementation in a proof of concept.

The research findings include a comprehensive list of requirements, including key requirements such as versioning, scalability, traceability, and data lifecycle management. The proof of concept demonstrated that the proposed design solution did not fully meet the requirements, indicating a complexity in addressing the problem beyond initial expectations. Due to time and resource constraints, a satisfactory full implementation of a proof of concept was not achieved. Moreover, a built solution meeting all the requirements to a satisfactory degree likely does not exist. Nevertheless, our research indicates that given additional time and resources, it is feasible to address the problem. Consequently, an interesting future work could be the development and implementation of such a solution.

Keywords: data management, training data, requirements, solutions, configuration management, versioning, machine learning, deep learning, management of data at scale

Acknowledgements

We would like to thank our supervisors, Lars Bendix and Fredrik Stål, for their continuous support and advice during this thesis. Especially Lars, who has provided us with many hours of advice, resources, and feedback – sometimes on a very short notice. We would also like to thank the Scandinavian Network of Excellence in Software Configuration Management (sneSCM), whose professional members gave us insights and suggestions that turned out to be very valuable.

Contents

1 Introduction **7**
1.1 Thesis Disposition 8

2 Background **9**
2.1 Precise Biometrics 9
2.2 Problem Statement 10
2.3 Research Questions 11
2.4 Methodology 13
2.5 Theoretical Background 14
2.5.1 Versioning 14
2.5.2 The Shared Data Problem and the Simultaneous Update Problem . 14
2.5.3 Private Workspaces and Merge Conflicts 15
2.5.4 Traceability 15
2.5.5 Data Lifecycle 15

3 Requirements **17**
3.1 Requirement Elicitation Process 17
3.1.1 General Requirements 20
3.1.2 Security Requirements 23
3.1.3 Other Requirements 24

4 Design **25**
4.1 Design Process 25
4.2 Design Solution 1: Binary Repository 28
4.3 Design Solution 2: Helix Core 29
4.4 Discussion and Evaluation of Design Solutions 30
4.4.1 Design Solution 1: Binary Repository 31
4.4.2 Design Solution 2: Helix Core 32
4.4.3 Choice of Design Solution for Proof of Concept 33

5 Implementation **35**
5.1 Proof of Concept . . . 35
5.2 What Worked? . . . 35
5.3 What Didn't Work? . . . 37
5.4 Discussion . . . 38

6 Discussion and Related Work **39**
6.1 Reflection on Our Own Work . . . 39
6.2 Validity . . . 40
6.2.1 Requirements . . . 40
6.2.2 Design Solutions . . . 41
6.2.3 Proof of Concept . . . 41
6.3 Generalizability . . . 41
6.4 Related Work . . . 42
6.4.1 Data Management Challenges for Deep Learning . . . 42
6.4.2 DataHub: Collaborative Data Science & Dataset Version Management at Scale . . . 43
6.4.3 Data Platform for Machine Learning . . . 43
6.4.4 Discussion . . . 44
6.5 Future Work . . . 45

7 Conclusion **47**

References **49**

Appendix A Interview Questions **55**
A.1 Pre-analysis Interview Questions . . . 55
A.2 Requirement Elicitation Interview Questions . . . 56

Appendix B Ranking of Requirements by Interviewees **57**

Chapter 1
Introduction

In the world of software development, management of source code is a well studied subject. Significant effort has been put into developing tools and practices to solve the challenges associated with version control, collaboration, and code quality assurance. The same can not be said about the management of data associated with the development process, such as for example training data for machine learning algorithms. In recent years, deep learning technologies have advanced rapidly, which has resulted in the launch of several new applications such as image generating models and chatbots, chatGPT being one of the more popular ones. Despite deep learning being almost a household name at this point, and the immense popularity of these applications, very little is known about how the companies developing these applications manage the vast amount of data that is required to create these models. There is also very little public research available on the subject of data management in a machine learning context.

This thesis was conducted at Precise Biometrics. A company that develops fingerprint matching algorithms for customers around the world. To achieve this they use machine learning algorithms in combination with conventional image analysis methods. As their number of customers have increased, the management of their training data has increasingly become a problem. Features like versioning, the ability to get an overview, automatic validation and seeing where in the data processing pipeline the data currently is are missing. The aim of this thesis is to identify the requirements for a training data management system and to suggest solutions that solve these requirements. More specifically, the research questions that this thesis aims to answer are as follow:

RQ1: How can a training data storage repository be designed to improve data management processes?

- **RQ1.1:** What are the requirements for such a training data storage repository?
- **RQ1.2:** What are some design solutions that fulfill these requirements?
- **RQ1.3:** Is the selected solution feasible and practical?

The research questions will be answered with the specific situation of the case company in mind, but the generalizability of the results is discussed in the Generalizability section in chapter 6.

To formulate and answer the research questions, this thesis was carried out in several phases. In the first phase, a pre-analysis was carried out, during which initial interviews were conducted with developers at Precise Biometrics to gain an understanding of the system and the underlying issues. Thereafter, the research questions were formulated. In the second phase, requirements were elicited and a comprehensive requirements specification was identified through in-depth interviews with developers, coupled with literature studies. In the third phase, two design solutions were theorized and in the final phase a proof of concept for one of the design solutions was implemented following consultation with the developers.

1.1 Thesis Disposition

The thesis consists of a background chapter which contains the background information on Precise Biometrics and how we arrived at the research questions, as well as the methodology of the study and some background theory. The results of the thesis are then presented in the three chapters Requirement Specification, Design and Implementation. The next chapter is Discussion and Related Work, in which generalizability and threats to validity are discussed, along with a comparison to related work and potential future research on the subject. Finally, the conclusion chapter presents the most important results and concludes how these can solve the original problem.

Chapter 2
Background

In this chapter, an overview of the case company will be presented, followed by an in-depth discussion of the initiating problem and motivation of why and how the research questions were formulated and derived from the initiating problem. Thereafter, the related theory pertinent to the initiating problem will be discussed, followed by a discussion of the methodology that will be employed in this thesis. The aim of this chapter is to a provide a broader context to the problem domain and motivate how and why we, the authors of thesis, formulated the research questions of this thesis. The current context and situation at the case company have played a significant role in shaping the direction and decisions made during the research process. Moreover, this chapter seeks to present the initiating problem in connection with the existing theoretical framework. This is done not only to position the study within the realm of existing knowledge, but also to present readers with the necessary theoretical context related to this research.

2.1 Precise Biometrics

In this section, a brief overview of Precise Biometrics, the case company where this master thesis has been conducted, will be presented to provide necessary context related to the background of this master thesis.

Precise Biometrics, which will be referred to as the case company in this thesis, is a digital identification software company headquartered in Lund, Sweden, with customers across the world. Their range of products include facial recognition solutions, visitor management solutions and fingerprint recognition solutions. This thesis research was conducted within the team responsible for developing fingerprint recognition solutions. This development team consists of approximately 20 developers divided between two locations: Lund and Shanghai. Precise Biometrics uses machine learning algorithms in combination with other conventional image analysis methods to provide tailored fingerprint recognition solutions designed to ad-

dress the specific needs and requirements of their clients. The fingerprint recognition models integrated into the product are configured using model-specific configuration files, which determine the training parameters, including the selection of training datasets and various other settings. These models are trained on different datasets, mainly provided by clients and are stored on a designated file server internally referred to as "the fingerprint database" (FPDB), which in addition to the training data includes additional necessary data, such as index files containing labels and metadata.

2.2 Problem Statement

In this section, we will discuss and elaborate on the initiating problem at the case company. This is important because by analyzing the current situation at the case company, we were then able to formulate research questions that addresses this initiating problem.

To formulate relevant and applicable research questions addressing the initial problem, it was necessary to investigate the current situation at the case company. This was achieved through conducting semi-structured interviews with several developers within the company, including both junior and senior developers. The rationale behind this approach lies in the potential for junior developers to offer different perspectives and discuss issues that may not be perceived by senior developers. Junior developers may also have a clearer recollection of the on-boarding process when they joined the development team, and they may remember which parts of the system initially seemed confusing to them. On the other hand, senior developers were interviewed because of their substantial experience in the domain of interest, which would allow them to provide valuable context concerning the present challenges and the current situation.

The responses from the interviews highlighted that the process of working with "FPDB" can be characterized as frustrating, complex, and quite messy. This primarily stems from the existing data storage system, which relies on a basic file server and suffers from numerous challenges, which will be discussed in this section.

Potential contributing factors to these challenges, as mentioned by some developers, include stress and deadlines. When developers are faced with deadlines and stress it is natural for them to allocate more time to complete tasks for customer delivery rather than addressing the system's challenges. Furthermore, the data management system has evolved organically over time. That is, when there were only a few customers, these issues were relatively inconspicuous, but as the number of customers has increased over time, accompanied by a corresponding surge in data volume, the deficiencies of the current setup have become more apparent. Additionally, a sense of familiarity with this system has developed over time, largely due to the inherent understanding of its operations by a small group of developers. Nonetheless, when viewed from an external perspective, such as that of a new employee, the system's efficiency appears to be compromised.

The first significant challenge with the current data management approach is *the inability to trace changes* made to files on the server and who made these changes. As a result, developers have had to manually generate various versions of the same file, sometimes resorting to arbitrary naming conventions to distinguish the files from one another, and to manually track these changes. This deficiency is a consequence of "FPDB" functioning solely as a file

server, lacking any versioning capabilities.

A second challenge arising from the current "FPDB" set-up is its *inefficiency*. Developers, depending on their experience at the company, frequently encounter difficulties in determining the purpose of files and their relevance, partly due to inconsistencies in the naming system. While there exist naming conventions, they are not consistently followed when making alterations or adding new files. Furthermore, the absence of a search mechanism for files or specific features within files, consumes a significant amount of time that could be more effectively allocated to other tasks.

A third challenge that has been identified pertains to the *limited knowledge* and understanding of specific sections of "FPDB", which are confined to certain developers who have been working at the case company for a longer period of time. This challenge could become more pronounced when an employee leaves the company, as their expertise in particular sections of "FPDB" may be lost, and new employees either have no familiarity with the file server or typically possess familiarity only with the portions of the file server they have directly interacted with. This issue arises from the sheer size and volume of the file server, the absence of any overview functionality to offer additional information of the structure of the file server, and, as previously mentioned, the inconsistencies in the naming system. Furthermore, the existing system's steep learning curve contributes to its inefficiency, demanding a substantial period for new developers to become familiar with it and understand it.

Considering all these challenges discussed above, there is a need for the case company to simplify their data handling process in order to enhance the effectiveness of their developers. The purpose of this master's thesis is, therefore, to investigate if there is a scalable solution for storing and managing training data used in different variants of machine learning models, which can be implemented by the case company.

2.3 Research Questions

The challenges discussed in the previous section illustrate the need for the case company to explore how they could better store, organise, and manage more effectively the training data and associated files which are used in different variants of machine learning models. As such, the research questions were designed to address some of the challenges of the existing data storage system and thereby, solve the initiating problem. The main research question **RQ1:** *How can a design of a training data storage repository be made to improve data management processes?* was formulated with the intention to deal with these challenges and investigate whether there is a way and how to improve the current data management process by finding a new design for the data management process.

We have structured the research questions as one main question, and sub-questions to help answer this main question. This approach aids in maintaining clarity and focus by breaking down the complexity of the main research question into smaller, manageable components. It also provides guidance for the research process. To answer the main research question, there is first a need to identify the requirements of a training data storage repository. A requirement specification is also important for developing the design solutions of a training data storage repository, and also helps to determine which design is the most suitable. With this in mind, the second research question **RQ1.1:** *What are the requirements for such a training data storage repository?* was formulated, followed by the third research question

RQ1.2: *What are some design solutions that fulfill these requirements?* to explore multiple design alternatives rather than focusing solely on one. To demonstrate and validate the feasibility of these designs, there is a need to develop a proof of concept. Therefore, once the potential design solutions have been developed, a proof of concept is intended to be implemented, which leads to the final research question **RQ1.3:** *Is the selected solution feasible and practical?*

- **RQ1:** How can a training data storage repository be designed to improve data management processes?
- **RQ1.1:** What are the requirements for such a training data storage repository?
- **RQ1.2:** What are some design solutions that fulfill these requirements?
- **RQ1.3:** Is the selected solution feasible and practical?

2.4 Methodology

In this section, we will discuss the methodology employed in this thesis, and motivate why the specific methodology was chosen in relation to the research questions. The planned work process for this thesis project is presented in Figure 2.1.

The process started with a pre-analysis phase, during which initial interviews were conducted to identify the initial problem and thereafter to formulate the research questions. The second phase was the problem analysis phase, which consisted of a literature study and in-depth interviews with the objective of identifying and eliciting requirements to be used in the design phase. In this report the result of this phase is presented in chapter 3: Requirements. The subsequent phase is the design phase, which entailed developing design proposals, followed by an evaluation of these design proposals. The results from this phase corresponds to chapter 4: Design. The final phase of the planned work process is the proof of concept phase, comprising the implementation of the proof of concept followed by an evaluation of the proof of concept. The results from the final phase are described in chapter 5: Implementation.

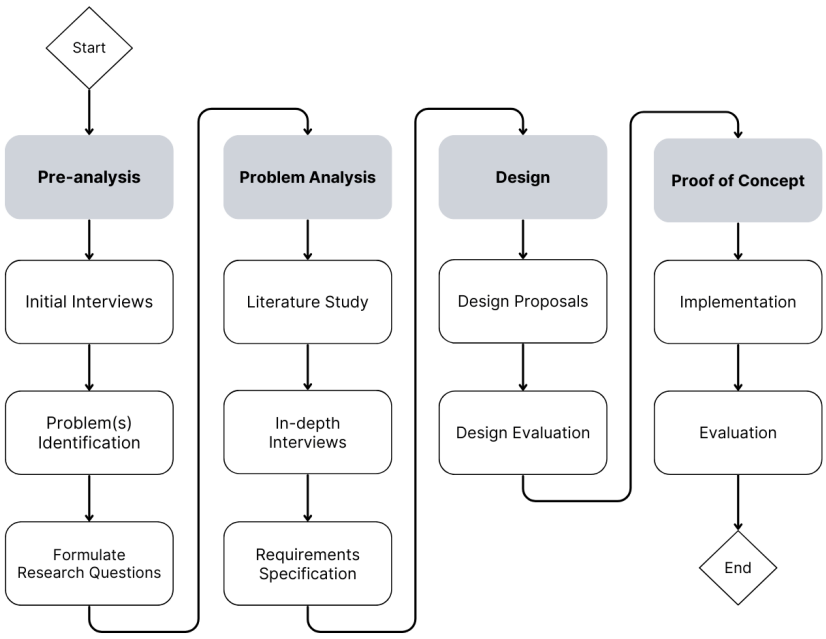


Figure 2.1: Planned work process.

The work process of this thesis project was inspired by the Design Research Methodology (DRM) [1, p. 113], which is why the thesis project was split into multiple phases as illustrated in Figure 2.1. The initial phase in DRM, referred to as Research Clarification, involves producing a goal document, which corresponds to the pre-analysis phase of this thesis project[1, p. 114].

In the second phase of DRM, referred to as the Descriptive study, the goal is to gain deeper insight into the problem at hand[1, p. 114]. In the context of this thesis project, the second phase entails the development of a requirement specification, which serves as the foundation for the design proposals. The primary objective of the second phase is to address **RQ1.1**. Our method for data collection during this phase is mainly through conducting interviews with

developers employed at the case company, in addition to a literature study. To discuss and evaluate the findings from the interviews, we conducted a focus group with developers and stakeholders at the case company. The intended goal was to formulate a detailed requirement specification. The requirement specification and a more detailed description of this phase can be found in the requirements chapter.

In the third phase, known as the Prescriptive Study in DRM[1, p. 114], the requirements identified in the prior phase will be used to develop design solutions addressing **RQ1.2.** In the final phase of this thesis, a proof of concept is implemented for one of the identified design solutions, aiming to address **RQ1.3**. Ideally, the proof of concept should be evaluated by developers as initially planned. However, due to the time constraint of this thesis and the limited resources of the case company dedicated to this thesis project, developers will likely not be able to evaluate the proof of concept.

2.5 Theoretical Background

In this section, we will discuss some relevant theoretical information pertinent to the problem domain of this thesis. The purpose of this section is to provide the reader with a foundational understanding of the theoretical concepts underpinning this thesis. These concepts are relied upon throughout this thesis, particularly when formulating the requirement specification and developing the design solutions.

2.5.1 Versioning

In Software Configuration Management (SCM), versioning is a central concept which refers to the ability to track changes. If a project is developed by a single individual in a strictly linear manner, versioning could be managed by simply saving the state of the project and tagging it with a timestamp. In many applications, such as Google Docs or Photoshop, this is done automatically and is just called the version history. When projects are developed by multiple people and versions start to diverge, however, things get a bit more complicated.

2.5.2 The Shared Data Problem and the Simultaneous Update Problem

The Shared Data problem is a typical SCM problem that means that changes made by one developer can interfere with the changes made by another[2, p. 10]. In the specific case of a machine learning project, it could for example mean that one developer can't run part of their pipeline because another developer is currently working on making changes to a dataset which is part of said pipeline, making the data temporarily corrupt. Another reason why one developer might be unable to do their work when another developer is making changes to a dataset is the Simultaneous Update Problem[2, p. 13]. The Simultaneous Update problem means that if both developers are making changes to a dataset, the developer that first saves their changes might have them overwritten by the second one. At the case company, these problems are rare because developers are assigned to different data sets from different customers and sensors. Because usage of such diverse data sets are rare, so are conflicts. In

the world of source code management this problem is mostly considered to be "solved" with the introduction of tools like git, which might lead to its oversight in other contexts.

2.5.3 Private Workspaces and Merge Conflicts

In SCM, the most common solution to the shared data problem is generally considered to be private workspaces. Copying the common baseline into a private workspace means that the developer is protected from changes made by others until a time of his choosing. Private workspaces do not, however, solve every problem. Every time a copy of a repository is made there is a risk of double maintenance problems, and private workspaces do not solve the simultaneous update problem. One solution to these problems is locking modules so that only whoever locked the module is allowed to make changes at a certain time. This is, in essence, how Precise Biometrics manages these problems today (although it is done manually, there is no actual lock), but the most common solution by far is to use merge conflicts to manage the simultaneous update problem and to manage the double maintenance problem through several other practices such as continuous integration and issue tracking/assignment systems.

2.5.4 Traceability

In software engineering, the term traceability refers to the ability to track, in both directions, changes through the different stages of development. For example, it could be the ability to link a specific requirement to a specific change in the code, to specific tests and further to a specific change in the end product. This is usually achieved through some kind of documentation, which could be created manually or automatically. In the case of this thesis, traceability will refer to the link between training data and what the training data is used for, for example machine learning models or validity tests.

2.5.5 Data Lifecycle

The data lifecycle and its management share similarities with the Software Development Lifecycle (SDLC)[3], a methodology familiar to most developers. SDLC refers to the development methodology where changes move through different stages such as for example development, testing and deployment. The data lifecycle in the context of machine learning refers to the various stages that data can traverse within a machine learning pipeline. Raw input data is typically distinguished from prepared training data, and prepared training data is not the same as validated and tested data that is "known to be good". At the case company, there is currently no system for managing the lifecycle of data.

Chapter 3
Requirements

Several challenges associated with the current data storage system and the process of working with it were identified in the problem statement. To address these challenges, this thesis seeks to explore how to improve the storage, organisation, and management of training data and associated files that are used in different variants of machine learning models at the case company. This chapter delves into the requirement elicitation process, comprising a literature review and interviews with developers in various roles at the case company, and it examines the elicited requirements aimed at addressing **RQ1.1:** *What are the requirements for a training data storage repository?* During the pre-analysis phase, we realised that addressing the initial problem necessitates, inter alia, an investigation to determine the requirements for an improved training data management system. These requirements are intended to be used as the basis for the design solutions that are developed in the next phase of this thesis project. This chapter presents the result from the problem analysis phase described in the methodology chapter.

3.1 Requirement Elicitation Process

The requirement elicitation process encompasses a thorough approach, combining a literature review, semi structured interviews and a focus group discussion meeting.

The process was initiated by deriving a list of potential requirements identified during the pre-analysis phase and in the literature found. We searched for literature through Google Scholar and LUBSearch combining different search terms such as "data management", "machine learning", "training data management", etc. Unfortunately we did not find a lot of related literature. One of the papers found at this stage is discussed in the related literature section[4].

Thereafter, a series of semi-structured interviews were conducted with developers, where questions were asked to acquire additional information regarding the current state, elicit additional requirements, and assess the derived requirements from the problem analysis phase.

A general template of the questions asked can be found in Appendix A, but the questions were developed and modified depending on who the interviewee was, when new requirements were discovered, and when our understanding of the requirements increased and definitions changed.

The interviews were held with a diverse group of developers, including junior and senior developers in various roles such as DevOps engineer, architect and product owner, as shown in table 3.1. This intentional diversity of the interviewees was aimed at gathering a comprehensive understanding of the system's needs from as many different perspectives as possible. The security manager at the case company was also interviewed in addition to the developers, with the objective of eliciting specific security requirements that were not attainable from other developers.

Table 3.1: List of interviewees.

Senior Developer 1	Product Owner
Senior Developer 2	Architect
Senior Developer 3	DevOps Manager
Senior Developer 4	
Senior Developer 5	Security Expert
Junior Developer 1	
Junior Developer 2	
Junior Developer 3	

To assess the derived requirements, all developers were asked to categorise them according to their importance from their perspective: "must-have" (highest priority), "should-have", "nice to have" (lowest priority), or "not needed". This approach was carried out in order to gain an understanding of the priorities of the development team at the case company. Following the conclusion of the interviews, a comprehensive analysis of all findings was conducted, and graphical representations of the results were generated (see Appendix B).

Subsequent to the series of interviews, a focus group meeting was conducted with five developers, including three seniors and one junior, in addition to the DevOps manager, with the purpose of discussing the results of the interviews and the additional requirements that were elicited during the interviews. The primary objective of the focus group was to discuss the results of the interviews, understand the developers' interpretations of the requirements and strive to achieve a common understanding. In the focus group, the participants were presented with the definitions of the different requirements that we had established so far as well as diagrams of how the developers had ranked the requirements during the interviews. Through discussion some of the definitions were changed and cemented and some misunderstandings were resolved. Extra attention was put towards the requirements where the developers had given different rankings during the interviews, so that a final decision about whether the requirement should be included or not could be made.

The resulting requirements were categorised into two groups: general requirements and security requirements. The reason for this is that the origin of the requirements are different. General requirements are derived from the interviews with developers and the literature study. In contrast, security requirements, were primarily derived from the insights gathered during the interview with the security manager at the case company, and weren't evaluated

by the other developers. In the following sections of this chapter, we will discuss the requirements we've elicited. They are listed according to their overall importance, each accompanied by a description of our definition and an analysis and motivation behind the requirement.

3.1.1 General Requirements

Requirement 1 - Versioning

The system must support versioning and enable the identification of who has made changes to the data, what data was changed and when the changes were made.

This requirement was first mentioned during our first talks with the case company, prior to the initial interviews in the pre-analysis phase. The results from the interviews and focus group shows that this requirement was deemed by an overwhelming majority of developers to be the most important requirement. As most interviewees mentioned this requirement themselves and almost everyone rated it as a must have. In the current implementation of "FPDB" there is no version control system, which leads to manual versioning of files by developers. It is not unusual to find two or more versions of a file in a location, with names such as "`old_filename.[fileending]`". This makes it harder to know what version of a file should be used, and makes it harder to get an overview of the data.

Requirement 2 - Scalability

The system must be able to handle significant increases in the size of stored data, types of data and number of files, as well as the number of users, without negatively affecting either its performance or the manual workload.

This requirement is another that was mentioned early on in the pre-analysis phase. One of the main reason this thesis came to be is that the current solution has proven to not be scalable. More precisely, the current solution worked fine when the number of customers and data was low, but as those numbers have increased, several new problems have appeared, as described in previous chapters. The definition of this requirement is not quite as clear as the other requirements as it's not a functional requirement, but we decided to define it as the first paragraph in this section, incorporating concepts like performance and amount of manual workload into its definition. It is also the hardest requirement to measure, since a complete implementation and extensive testing would be needed in order to completely assess its fulfillment. Most developers rated scalability as a must have.

Requirement 3 - Traceability

The system should support the capability to trace what data was used in a specific model or process, and the other way around.

Traceability is a requirement that was originally inspired from previous experience in Software Configuration Management, where the term traceability usually refers to the ability to trace features and changes though the different stages of the software development life cycle, stages like requirements, development and testing. Since this thesis is about managing training data, the term traceability refers to the connection between the data and the specific model or process it was used in. The current system used by the case company already supports traceability to some extent, it is possible to discern what index files were used in

which trained model. However, since the index files are not versioned, traceability in the direction from model to data is fundamentally broken. One developer mentioned that he was tracing his own work manually, and similarly another developer mentioned that traceability was possible with the current setup but that it required extra manual work. This is also something that is noted in related works[5]. Traceability in the direction from data to model or process is not present at all, but the interviewees did not seem to think that feature was very important. In general, the interviewees found traceability to be relatively important, and were split between rating traceability as a "must have" or a "should have".

Requirement 4 - Data Lifecycle Management

The system should provide the capability to determine the current stage of the data. For example, it should be able to identify whether the data has recently been received from the customer and is awaiting processing, whether the data is currently in an active state of being worked on, or if it has already been integrated into production or is part of a legacy product.

This requirement is also inspired by previous experience in Software Configuration Management. In software development, changes or new features could, for example, start in a requirements phase, move on to a development phase, into a testing phase and end up in a deployment phase. This is called the software development lifecycle[3]. The idea is the same for the data lifecycle. Several interviewees expressed that a basic level of data lifecycle management was needed, especially the ability to distinguish raw data from processed data that can be used in the production environment. The interviewees were split between rating this requirement as a should have and a must have.

Requirement 5 - Overview Functionality

The system must provide the capability to obtain a comprehensive overview of the content within various data sets.

An overview of data was something that was also mentioned very early in the initial interviews. The interviewees expressed that looking through the data required a lot of manual work, navigating through the file structure and opening different files to find out what type of data a specific data set contained. The interviewees rated this requirement as a should have.

Requirement 6 - Avoiding Shared Data/Simultaneous Update Problem

The work of one developer should not affect anyone else, and that developer should not be affected by anyone else's work, unless they want to.

The shared data and simultaneous update problems are established problems in software configuration management[2, p. 10]. During the in depth interviews, it was something that many interviewees downplayed, which surprised us, especially since one of the interviewees

was quoted as saying “I can’t run my pipelines right now, because [another developer] is cleaning up the database.”. The explanation we got was that it was pretty rare for two developers to be working on the same part of the FPDB, and that they avoided conflicts by communicating. For this reason most of the developers rated it as a nice to have. However, when discussed at the focus group, the attendees agreed that it was an actual problem. It was deemed that some developers had assumed that this was already a part of the versioning requirement, or perhaps not understood the concept fully, and it was decided at the focus group meeting that the requirement was still something that should be included in the requirement specification.

Requirement 7 - Automatic Validation

The system must support data validation in a continuous manner or upon any occurrence of changes being made.

This requirement was something that appeared in the initial interviews, and is also something that is common in modern software development. A script for validating the index files already exists, but it is only run manually, and there is no reliable way to immediately tell if a file has passed validation since the last time it was changed. Similarly to when discussing the data lifecycle management requirement, some developers expressed that it would be useful to know if data is ready to be trained on. Opinions differed on this requirement, but half of the developers ended up rating it as a must have.

Requirement 8 - Searchability

The system must support the capability to search for specific files or features within the data, such as, for example, "dry fingers" from a particular customer from a specific year.

Searchability was something that was mentioned in both the initial and the in-depth interviews. Interestingly, this concept seemed to be very important to some interviewees and not very important to others, probably because of the different work duties of different developers. One developer said “If I want to find fake fingerprints made by latex fingers, I have to manually look for them, or write a python script to find them for me”. The interviewees were split fairly evenly between rating this requirement as a must have, should have and nice to have.

Requirement 9 - Load Only Required Data

FPDB contains terabytes of data and thousands of data sets. The largest data sets contain more than a million individual files. It is not feasible to have a local copy of the whole repository, so only the necessary files should be loaded onto the developers computers.

FPDB is several terabyte big and contains millions of files. Because of the size and continuous growth of FPDB, it’s not possible for all the data to be stored on each developer’s computer. There’s also no need to check out all of the data at once, since each developer only works on one or a small number of data sets at a time. During the design phase of the project, we realized that we were already working towards this requirement that we hadn’t

formulated yet, so it was added to the list of requirements. As an example, a distributed version control system like git would not be an acceptable solution without some additional functionality, since git stores a complete copy of the repository on the computer. Since this requirement was added at a later stage, its importance was not rated by the developers, however, we consider it a must have.

3.1.2 Security Requirements

The security requirements mainly deal with the sensitivity of the type of data, which, in this case, is biometric data, and is thus subject to regulatory scrutiny. Furthermore, since the case company serves clients in various regions, including Asia and North America, the data management and storage methods carry legal implications. Thus, emphasizing the data's source and collection location is crucial in light of these regulatory considerations. Additionally, because the case company is headquartered in Sweden and operates in the EU, EU regulations, such as GDPR laws, are applicable. The security requirements are discussed and presented below, in no particular order.

Requirement 10 - Local Server Storage

The system must store data on a local server, rather than on the cloud due to regulatory considerations.

A previous investigation had been carried out at the case company to assess the feasibility of using a cloud-based system as a platform for storing biometric training data. The conclusion reached from that investigation is that local storage is more appropriate due to regulatory issues from having clients in different regions of the world. Consequently, it is highly improbable that the case company will move the data to a cloud-based storage system in the future. In short, there are no technical reasons not store the data on cloud-based storage systems, except for regulatory.

Requirement 11 - Encrypted Storage

The system must encrypt all data to ensure the security and confidentiality of the data.

There are two main reasons why using encrypted storage is desirable, especially when taking into account the sensitivity of the data. The first reason is confidentiality assurance, to protect the confidentiality of data and prevent unauthorised individuals or entities from accessing sensitive information. The second reason is to ensure the protection of intellectual property since the data is owned by clients.

Requirement 12 - Ability to remove personal data

The system must support the removal of data upon the request of its owner.

Compliance with GDPR regulations in the European Union, as well as other regulatory requirements in various regions, necessitates the capability to erase personal data. This erasure process should ensure that personal data is fully deleted and not retained in backups or

version history. Since the data is not owned by the case company, there must be a process to enable the deletion of data.

3.1.3 Other Requirements

More requirements were presented to the developers, including built-in support for metadata, identification of outdated or garbage data, and automatic deployment; however, most interviewees agreed that they were not ultimately necessary for the solution.

Chapter 4
Design

Following the completion of the requirement elicitation process, which, as explained in the previous chapter, form the foundation for the intended design solutions, the design phase constitutes the next stage in addressing the initial problem, and aims to answer **RQ1.2** *What are some design solutions that fulfill these requirements?*. Therefore, this chapter is dedicated to investigating and exploring potential design solutions aimed at addressing **RQ1.2**, discussing how these design solutions fulfill the requirements, in addition to reflecting upon the designs' respective strengths and weaknesses in relation to the initial problem.

This chapter begins by discussing the design process, followed by investigating and exploring two design solutions and lastly, motivating the design solution chosen for implementation in a proof of concept. This chapter may serve as a valuable starting point for those interested in implementing a similar solution addressing a related problem.

4.1 Design Process

The design process began with an initial step of investigating academic papers and studies related to the problem domain, with the intention of identifying an academic framework which could be used or built upon in our design solutions. The investigation, however, revealed a scarcity of academic research pertaining to the problem this thesis aims to solve. As a result, an alternative approach was adopted, focusing on exploring available tools used by the industry.

Therefore, the next step was to compile an extensive list of potential tools capable of addressing as many requirements as possible. During the early stages of this thesis, a meeting was organized at the case company in collaboration with the Scandinavian Network of Excellence in Software Configuration Management (sneSCM) [6], with the aim of gathering insights from industry experts and veterans. The primary purpose of the meeting was not explicitly to discuss potential design solutions, but rather to discuss the broader topic of this thesis and the initial problem. Nevertheless, discussions emerged about various tools that

could potentially address parts of the identified problem, providing valuable insights that we were now able to investigate in the ongoing design phase.

As previously mentioned, due to the absence of relevant academic research, the design process was oriented towards specific tools that held potential to fulfill the requirements. These tools were theoretically investigated through reading documentation. Numerous tools were investigated (see Table 4.1), but owing to the time constraints in this thesis, a decision was ultimately made to focus only on a few alternatives. These alternatives were selected for further detailed exploration as they were perceived to possess the greatest potential in addressing the majority, if not all, of the requirements. Each alternative was then compared against all requirements. Combinations of several tools were also considered but no obvious satisfying combination was found. Consequently, the two final design solutions explored in detailed in the following sections in this chapter is attributed to this decision-making process. As can be seen in Table 4.1 and as mentioned in the literature [5] [4], it's clear that most of the publicly available tools are more focused on training and evaluation than on managing data.

Table 4.1: List of all investigated tools.

Alectio [7]	Discarded. The tools seem focused on optimizing datasets and labeling as well as monitoring the performance of models.
Aquarium [8]	Discarded. This tool is focused on providing visualization and data analysis, which is not the main focus of this thesis.
Artifactory [9]	Fulfills several requirements, part of design solution 1.
Data Version Control (DVC) [10]	Documentation was thoroughly studied but the tool was not included in the design solutions. This tool can potentially solve several requirements, such as versioning and data lifecycle management, but it would need to be combined with other tools that can solve traceability, searchability and overview functionality.
Eiffel [11]	Discarded. Even though the traceability is mentioned, the project doesn't seem to be about data management at all.
GitLFS [12]	Documentation was thoroughly studied but the tool was not included in the design solutions. Similar reasoning to DVC. There is also related research outlining performance issues in git-based systems in this context.[13]
Helix Core [14]	Fulfills several requirements, part of design solution 2.
Hydra [15]	Documentation was thoroughly studied but the tool was not included in the design solutions because it potentially only fulfills few requirements. It could be considered for future research.
Labelbox [16]	Discarded. This tool doesn't support on-premise storage. For another situation, this tool might have some potential.
LatticeFlow [17]	Discarded. The purpose of this tool appears to be diagnostics, to fix erroneous data and improve model performance.
Openlayer (FKA Unbox) [18]	Discarded. The tool is mainly focused on evaluation and monitoring of ML pipelines, not data management.
Sonatype Nexus Repository [19]	Similar to Artifactory, considered for design solution 1.

4.2 Design Solution 1: Binary Repository

The first design solution is a solution based on the possibilities of binary repository managers such as Artifactory [9] or Sonatype Nexus Repository [19]. Binary repository managers are generally used to manage binary dependencies and releases in the form of compiled executables. In theory, training data in the form of images is a type of binary dependency, so the idea is that a binary repository should be well suited to manage the training data. The solution is idealized to some extent, since the capabilities of different binary repository managers can differ and since it's not clear if all aspects of the solution are technically feasible without building and testing the solution.

Setup

Because of the lack of version control functionality in binary repositories, in this solution the index files containing labels and metadata are not managed in the binary repository, but instead moved to the code repository, where they are managed in git. This means that either the index files need to be submitted separately from the data, or, perhaps preferably, the index files need to be separated from the data somewhere in the pipeline. Managing the index files in git will ensure that the versioning requirement is fulfilled and that the requirement to avoid shared data/ simultaneous update problem is at least partially fulfilled. The perfect solution would of course be if the binary repositories had better versioning capabilities, but currently it appears that the way version control is managed in these programs is by simply attaching a revision number to the configuration item, which is not significantly different from the current versioning system at the case company, which relies on file names. Making the index files version controlled completes the case company's traceability chain in the direction from model to data. Since binary repositories are also intended to manage builds (equivalent to models in this case), there is also potential to expand the traceability to two directions in the future.

In order to fulfill the data lifecycle requirement, the binary repository is set up to have multiple stages which the data can move through. Some of the interviewees had previously expressed that there was a desire to empower the customers to upload and validate their own datasets. Because of this, the initial stage could be set up to be accessible for both customers and developers at the case company. Additionally, there could be a second stage where the data is worked on before deployment, and a final stage where the data is moved once it is ready for deployment. The exact stage configuration is not set in stone and should be modified to fit the needs of the users.

In order to fulfill the automatic validation requirement the stages need to be set up with different validation tests, so to change data or move it from one stage to another, some tests need to pass. The tests should be different depending on the stage. For example, the initial stage should have no or few tests so that the system does not prevent the customer from uploading their data in a case where the customer encounters a problem which they are unable to fix themselves.

Some binary repositories provide metadata functionality, called properties in Artifactory. Different properties can be assigned to data sets. The properties could specify different information about the data, such as which customer, sensor or condition the data is produced from. These properties can be used to search through the data and could potentially

also be used to construct a graphical interface, which could help give the developers a better overview of the data.

A binary repository can be mounted on a local server. It is easy to pick and choose what data is loaded onto the developers computer, fulfilling several requirements. Since binary repositories do not have any advanced versioning capabilities, and since the index files which are stored in git are completely anonymized, it's also possible to completely remove any personal data upon request. When the designs were compared to the requirements this solution was able to fulfill 7 out of the 12 requirements completely, with versioning, overview and avoiding shared data/simultaneous update problems being partially fulfilled. It is not clear if the design fulfills the scalability requirement. The binary repositories do not appear to support encryption of the data by themselves, so that requirement would have to be achieved in some other way, possibly by encrypting the data before upload or by making sure that the servers that the binary repository is run on has some form of built in encryption. Since there are likely other ways to achieve this requirement, and because the requirement is not a very high priority for the case company, encryption of data is not a part of this design solution.

4.3 Design Solution 2: Helix Core

This design solution is based on Helix Core, developed by Perforce (also formerly known as Perforce), which is a version control software for large scale development environments and which supports the versioning of binary files. The basic idea behind the Helix Core version control system is that it uses client-server architecture to implement version control management[20]. Shared file repositories that contain every revision of every file are managed by the Helix Core server, and files are organized into directory trees[20]. Additionally, a database is maintained by the server to track data associated with files and client activity, such as logs, user permissions, metadata, configuration values, etc[20].

Setup

Technically, the Helix Core server would be installed on the local machine where the biometric data and index files are stored[21]. This design solution would therefore keep the current arrangement of having index files and biometric data stored on the same server since both can be versioned with Helix Core, and therefore there is no need to separate them. The index files and biometric data would be organized into directories. To ensure consistency, any newly created directory must adhere to the naming convention, but since Helix Core cannot enforce this, the development team must commit to following these naming conventions. One way developers can access the server, is through the Helix Visual Client, which provides a comprehensive overview of all the server's content[22].

The versioning requirement is fulfilled by Helix Core since it versions every type of file, even binary files[23]. Both biometric data files and the index files would therefore be versioned. Each time a file is changed, an updated file is submitted to the server as a new revision. The version control management system in Helix Core also fulfills the requirement of avoiding shared data/simultaneous update problem, as Helix Core synchronises the files in the master repository to local workspaces. Since files are by default in read-only state, changes made to files are made on local workspaces. Only after the files have been checked out,

can developers make changes and submit them back to the master repository. Furthermore, similar to git, Helix Core also has a conflict resolution mechanism to resolve any potential conflicts[24].

The traceability requirement, i.e. that system should support the capability to trace what data was used in a specific model or process, and the other way around, is not fulfilled in this design solution. Though Helix Core provides an immutable historical record of all the changes ever made to a file in the repository, and by tracking all changes made to files any changes to the files can be traced back to the person who made the changes, they cannot be tracked to the specific model or process in which they have been used.

This design solution has the potential to meet the scalability requirement, which entails the system's ability to adapt to significant increases in data volume and types without compromising performance or increasing manual workload. Currently, the file server 'fpdb' holds approximately 4TB of data. According to Perforce, Helix Core is designed to handle tens of millions of daily transactions and petabytes of data [23]. Since the needs of gaming development companies, whose files may amount to hundreds of terabytes or even petabytes of data, can be accommodated by the Helix Core Server, it is also possible for the data of the case company to be handled by Helix Core, and for it to manage future storage and management as data expands [25].

One way in which the data lifecycle management requirement could be fulfilled is by utilizing separate directories for the data with access permissions tailored to the accessing party. Similar to in the binary repository design, three stages could be represented by different directories: a directory for the unvalidated stage, to which customers also have access and can upload their data; a separate directory for the validated stage; and an additional directory for the processed stage, both of which only developers would have access to. Another way could be by creating and using branching.

The requirement of overview functionality can be fulfilled by the Helix Visual Client (P4V), a desktop application that grants access to versioned files in Helix Core through a graphical interface [22]. Through this interface, a comprehensive overview of the content within various data sets can be achieved. By integrating with Helix Visual Client, the searchability requirement could also be fulfilled.

The requirement of automatic validation, i.e. that the system must support data validation in a continuous manner or upon any occurrence of changes being made, is not fulfilled in this design solution using HelixCore as a main tool. It would most likely need a manual implementation and integration of validation tests for example. As for the requirement of encrypted storage, secure communication between clients and servers is guaranteed with Helix Core according to Perforce [26]. The last requirement, which is the ability to remove personal data, is also fulfilled.

4.4 Discussion and Evaluation of Design Solutions

In this section, we will evaluate and discuss the design solutions, considering their strengths and weaknesses. In general, the evaluation of the design solutions rely on their theoretical capabilities. To thoroughly evaluate the design solutions regarding their effectiveness in meeting the requirements in practice and ease of implementation, thorough testing is de-

sired. However, owing to the time limitations of this thesis, we were only able to conduct thorough testing on one of the two design solutions, which will serve as a proof of concept (See Chapter 5). At the end of this section, we will discuss and motivate the selection of the design solution chosen for implementation in the proof of concept.

4.4.1 Design Solution 1: Binary Repository

Even if all aspects of this design proposal are technically achievable, there are still some challenges that remain unsolved. Some of the data in the index files act both as labels and as metadata which provides information and searchability for the developers. For example, whether or not a finger has the "latex" tag can be relevant both for the developer, who might be working on a spoof detection algorithm and needs data to test it, and for the systems that are using the data to train or validate ML-models. Moving the index files to the code repository and adding properties to the data sets solves the versioning problem but leads to a double maintenance problem (information such as "latex" could exist both in the index file and as a data set property).

Using this solution, the shared data and simultaneous update issues are mostly solved, since the index files are now managed in git once they have been passed the validation phase. But it is unclear how the system could handle two different users trying to move data through stages at the same time or making changes to the index file before it moves to the git repository at the same time etc. Other potential problems with the solution are performance and price. As the solution has not been tested on large scale data sets, it is possible that there are performance issues in the solution. When it comes to pricing, both Sonatype Nexus and JFrog Artifactory have free and open source versions, but these versions might be lacking some desired features, such as tool integration and authentication options. Depending on the need for these features, the financial cost of the solution could be high.

Table 4.2: Summary of requirements fulfilled by design solution 1.

Req 1 - Versioning	Partially fulfilled.
Req 2 - Scalability	Potentially fulfilled.
Req 3 - Traceability	Fulfilled.
Req 4 - Data lifecycle management	Fulfilled.
Req 5 - Overview functionality	Partially fulfilled.
Req 6 - Avoiding shared data/simultaneous update problem	Partially fulfilled.
Req 7 - Automatic validation	Fulfilled.
Req 8 - Searchability	Fulfilled.
Req 9 - Load only required data	Fulfilled.
Req 10 - Local Storage	Fulfilled.
Re1 11 - Encrypted storage	Not fulfilled.
Req 12 - Ability to remove personal data	Fulfilled.

4.4.2 Design Solution 2: Helix Core

The primary advantage of this design solution lies in its ability to meet the majority of the requirements, particularly those ranked highest. However, its main weakness is its potential cost, which may present a challenge for any development team or company seeking to implement it. Naturally, this is a subjective matter because it depends on the financial capacity and priorities of a company. Another weakness is that while it fulfills most of the requirements, there are some requirements that it doesn't naturally fulfill as they require manual implementation.

One of the challenges we identified in the pre-analysis phase and that are mentioned in the problem statement of this thesis, is that as a result of the inability to trace modifications made to files on the server and who made these changes, developers have sometimes resorted to arbitrary naming conventions to distinguish the files from one another. Additionally, there are also inconsistencies in the naming system, as naming conventions are not consistently followed. In this proposed design solution, index files and biometric data would be organized into directories, which would necessitate the renaming of directories according to a specific naming convention. Helix Core lacks the capability to enforce this, which would require the development team to commit to adhering to the naming conventions. However, while this design solution does not directly address the challenge of consistently enforcing naming conventions, it does fulfill the requirement for versioning. This might facilitate this process by encouraging developers to comply with naming conventions due to the increased transparency provided by versioning features.

Another insight we've gleaned from this design solution is that, while this design solution is intended to be implemented in a way that resembles the current setup of having the index files and biometric data stored on the same server, it can also be configured to function as a centralized repository. This means storing all digital assets of the case company, including the source code, data, index files, and other digital assets, in one place, as Helix Core appears to be scalable and was built to handle tens of millions of daily transactions and petabytes of data. A centralized repository can be utilized as a centralized location for the management and provision of master data to the organization [27]. This facilitates the consolidation and integration of data from diverse sources into a unified instance and representation for each distinct master data object [28]. However, it should be noted that a centralized repository is likely not considered ideal for the case company and the development team at this time for practical reasons, as it would require the entire development environment to be changed.

Table 4.3: Summary of requirements fulfilled by design solution 2.

Req 1 - Versioning	Fulfilled.
Req 2 - Scalability	Potentially fulfilled.
Req 3 - Traceability	Partially fulfilled.
Req 4 - Data lifecycle management	Fulfilled.
Req 5 - Overview functionality	Fulfilled.
Req 6 - Avoiding shared data/simultaneous update problem	Fulfilled.
Req 7 - Automatic validation	Not fulfilled. Requires manual implementation.
Req 8 - Searchability	Fulfilled.
Req 9 - Load only required data	Fulfilled.
Req 10 - Local Storage	Fulfilled.
Re1 11 - Encrypted storage	Fulfilled.
Req 12 - Ability to remove personal data	Fulfilled.

4.4.3 Choice of Design Solution for Proof of Concept

The choice of the design solution to be implemented in a proof of concept was influenced by a few considerations. The first is the input provided by the case company based on their priorities. The second is the time constraints of this thesis and our own assessment. The feedback from the case company suggests that implementing the helix core design solution would be financially impractical for them at present or in the near future. Consequently, Design Solution 1: Binary Repository was considered to be more suitable from a financial standpoint. Moreover, it is perceived that a binary repository would be easier to test at the case company. Considering the time constraints of this thesis project and the limited resources provided by the case company, our assessment indicates that thoroughly testing and implementing a proof of concept for design solution 1 would require less time compared to design solution 2. As a result, we conclude that design solution 1: Binary Repository is the preferred option for implementation in a proof of concept.

Chapter 5
Implementation

The design solution chosen for implementation in a proof of concept is design solution 1: Binary Reposiory. In this chapter, we will discuss and analyze the implementation of the chosen design solution in a proof of concept and discuss the limitations and challenges that were encountered during implementation. The purpose of a proof of concept is to demonstrate the feasibility and practicality of the design solution, therefore this chapter aims to answer research question **RQ1.3:** *Is the selected solution feasible and practical?* This chapter is divided into two sections, the first explores the implementation, what worked, and what didn't work, and the second discusses the overall implementation.

5.1 Proof of Concept

To get started, JFrog Artifactory was installed on a local server at the case company [29]. As this is a proof of concept implementation, we were working in a testing environment, and not on the actual machine currently hosting the data. Test data was subsequently uploaded through the graphical interface. In the following two sections, we will discuss what worked, and what didn't work according to the expectations of the design solution.

5.2 What Worked?

Local Storage

Artifactory was installed on a local server and test data was uploaded with no issues.

Traceability

As mentioned in the design section, the case company already has some flawed in-house functionality for traceability in the direction from the model to the data, which relies on

the index files. Since the index files can change and are not version controlled, the chain is incomplete. This problem is fixed by moving the index files to a version control system, in this case git. It should be noted, however, that Artifactory is designed to manage binaries of all types, including builds and releases. It's therefore possible that Artifactory has a functional system for bidirectional traceability. However, this was not investigated as the case company did not indicate any intentions to alter their release management process and was satisfied with traceability being limited to one direction.

Data Lifecycle Management

There are different ways to organize the data in Artifactory. We suggest that each customer is represented by a project, and in each project, the customer's different data can be added in the form of repositories. This approach ensures clarity and makes use of Artifactory's built-in functionality, particularly since projects can be assigned different user permissions.

To represent the different stages the data should move between, the environments feature in Artifactory can be used to assign each repository a stage. There are already two default environments in Artifactory, DEV and PROD, with the feature of adding custom ones, depending on the number of stages the development team would want the data to move between [30]. To assign or change the environment of the data, developers can select the desired environment in the interface, and it will be visible to all users who have access to the project. Unlike the current setup at the case company, where the status or stage of the data isn't directly visible in the file server (or whether it's been cleaned and prepared for training), Artifactory's interface enables users to determine what stage the data is currently in. This fulfills our requirement of data lifecycle management.

Overview Functionality

Through the user-friendly interface, developers can gain a comprehensive overview of the different datasets, repositories and projects. Additionally, Artifactory includes a 'properties' feature that allows developers to "tag" repositories with keywords, which enables developers to see additional information about the repositories.

Searchability

The interface of Artifactory includes a search bar, which enables developers to browse the different projects and repositories. However, it is uncertain whether this requirement is fully met, as the effectiveness of this feature largely depends on the utilization of the properties function to fulfill the requirement effectively.

Load Only Required Data

Artifactory is a centralized repository and a user can download whichever data he or she desires.

Simultaneous update

To replicate the simultaneous update issue, we used two different logins to access the interface and perform different actions at the same time. We found no apparent issues with this.

Ability to Remove Personal Data

Since Artifactory does not have any advanced versioning functionality, completely removing data is easy. Backups of data should also be taken into account, but backups were implemented in this proof of concept. The index files which are versioned in git are anonymized and do not contain any personal information.

5.3 What Didn't Work?

Versioning

Although versioning is fulfilled by moving the index files to the source code repository, versioning is not fulfilled in Artifactory. As mentioned when discussing the data lifecycle management requirement, there is no way to see who moved data from one stage to another, when the data was moved, or connect the move to any specific changes. This means that the requirement for versioning, the way we defined it, is not fulfilled.

Scalability

The scalability of the system remains unknown. In order to test the scalability and reach a proper conclusion if it is scalable or not, we would ideally need to move a large amount of data (if not the entire data storage) to the binary storage, and test the system to verify that increasing the number of files, users and total data volume does not negatively affect the performance or the manual workload.

Encrypted Storage

Artifactory does not support the ability to encrypt the data. Data encryption needs to be achieved some other way, possibly by using drives with built-in encryption for the server or encrypting.

Automatic Validation

When this design solution was envisioned, we hoped that it would be possible to set up tests to validate data when they moved through the data lifecycle stages, however, it does not appear that Artifactory has this functionality. We were also unable to find a way to automatically validate files that were uploaded.

5.4 Discussion

During the implementation of the proof of concept, it was clearly noticeable that JFrog Artifactory was not designed with the aim of managing training data. The main focus of Artifactory is the integration with other tools, such as build tools and dependency managers, and much of the required functionality would have to be built on top of Artifactory using the command line interface or API. Nevertheless, we found it to be able to fulfill some of the requirements, as we have discussed.

It should be noted that some of the functionality that one might expect from a lifecycle management system is missing, such as validation tests or reviews, the ability to connect a move to specific changes that have been made, determining who moved the data from one stage to another or determining when the data was moved. If one is used to source code management tools such as Git in combination with GitLab or GitHub, this is most likely a disappointment. In hindsight, our definition of data lifecycle management was probably too narrow, however, we decided that some of this missing functionality fits with our definition of automatic validation and versioning.

Chapter 6
Discussion and Related Work

In this chapter, we will first reflect upon the methodology and work process of this thesis. Thereafter, we will discuss the results, consider potential threats to validity and the generalizability of our findings. Subsequently, we'll examine related work and, finally, address future work prospects. This chapter provides insights to readers into the challenges we encountered during our research. It is also important because it discusses the applicability of our results to other companies and software development teams. Furthermore, by examining related work, we discuss how the findings of our research intersect with existing literature, thereby providing a broader context for understanding. Moreover, it outlines potential directions for future research, providing interested readers with opportunities to explore further extensions of our research.

6.1 Reflection on Our Own Work

In this section we will reflect upon the work process of this thesis project. We will discuss the aspects we found effective and positive, as well as those that presented challenges. We will also reflect upon areas where improvements could be made, and discuss what we would do differently in future projects. The purpose of this section is to present readers with an understanding of the challenges encountered during this research, and provide insights and lessons that can be drawn from this study for consideration when conducting similar research.

Dividing the work process of this thesis into phases, as illustrated in Figure 2.1, proved to be a good decision. Since the research questions naturally unfolded in a chronological order, they constituted separate phases in the work process. This approach felt not only logical but also highly effective. It provided a clear and structured framework for conducting our research, allowing for a systematic exploration of each question and guiding the research towards its intended objectives. Furthermore, we are particularly satisfied with the requirement specification and the elicitation process behind it, since this is the part of the thesis we believe to be the most significant and generalizable, and the part where we have the most

data to support our results.

One of the most time-consuming and challenging aspects we encountered in this thesis project was determining the initial problem and formulating the problem statement. Initially, the problem this thesis aimed to address lacked clarity and precision. The case company was aware that their current process of working suffered from issues, but not exactly what the issues were. This lack of insight necessitated a significant investment of time to refine and delineate the problem. Undoubtedly, this aspect was of immense significance given its pivotal role in shaping the subsequent phases of the thesis project. A clearer problem statement from the case company prior to initiating this thesis project would have greatly facilitated this process and could have allowed for redirecting more time and effort towards other aspects of the project, such as exploring additional tools during the design phase, thoroughly evaluating the design solutions or implementing a second proof of concept.

The requirement elicitation process also proved to be a more time-consuming process than we expected. This is largely due to our desire to elicit and include as many requirements as possible. This, however, presented a challenge during the design phase. In hindsight, focusing on the most important requirements would have been more effective and would have allowed for greater flexibility during the design phase. The approach adopted in this thesis resembles the waterfall model, wherein we completed one phase before beginning another. In future projects, we would instead employ a more agile approach with an iterative process. This approach could potentially have allowed us to try out more solutions practically at an earlier stage in the process and accommodate changes in the requirements during the work process when new deficiencies were found in the tested solutions.

In future projects, we would also opt for breaking down requirements into smaller, more easily manageable requirements. For instance, the versioning requirement could be divided into three smaller requirements: identifying the user responsible for changes to the data, identifying the data that was changed, and recording the timestamp of when changes were made. This approach would not only enhance clarity, but also mitigate the complexity of addressing broad requirements, as working with broader requirements was at times frustrating and required constant recollection of the entire definition.

6.2 Validity

In this section, we will address the threats to the validity of the results presented in this thesis. This discussion is essential as it enables readers to judge the validity of the findings in light of the identified threats. We will address the validity threats associated with the results for each research question separately in the following three sections.

6.2.1 Requirements

One potential threat to validity is selection bias of the interviewees. To mitigate this potential threat and enhance the diversity among interviewees, we chose to interview eight developers with different roles and experience within the development team. We also employed a ranking system to gain a comprehensive understanding of the most prioritized needs of the developers, and thereafter, we conducted a focus group to evaluate the findings. While there may be potential differences in opinions compared to developers from other compa-

nies or development teams, the findings from the literature review indicate that the results of this research closely resemble those highlighted in the existing literature, thus strengthening the validity of the result. Therefore, we contend that the requirement specification offer a conclusive result.

6.2.2 Design Solutions

A potential threat to the validity of the result is the lack of evaluation of the design solutions by the development team, mainly due to the time constraints of this thesis. The design solutions primarily rely on the author's previous experience and theoretical knowledge of the field of configuration management, both having taken the configuration management course by Professor Lars Bendix at LTH. To mitigate this threat, our approach was to assess the design solutions to determine which requirements were met and which were not. However, a comprehensive evaluation of the design solutions by the development team could strengthen the validity of the result.

6.2.3 Proof of Concept

The proof of concept was conducted in a testing environment, with limited data used as testing data. While some of the requirements were fulfilled, some requirements were not. It is possible that had the proof of concept been conducted using a significantly larger volume of data, it might have allowed for a more comprehensive evaluation of requirements, including the scalability requirement. This could potentially enhance the validity of the results. Nonetheless, we argue that while the proof of concept results are inconclusive, they provide valuable indications.

6.3 Generalizability

In this section, we will discuss the generalizability of the results, that is, to which extent they are specific to the case company, and to which extent they are applicable to other organizations that use data at a large scale in their development process. This section is important as it assesses the potential for other organizations to use or draw inspiration from the results of this research.

We consider the requirements to be the most important result of this thesis. In terms of generalizability, we posit that the requirements of versioning, scalability, traceability, data lifecycle management, searchability and overview functionality are applicable to most organizations and development teams in similar situations where data is used at a large scale in their development process. The reason for this is the overwhelming consensus that existed among those interviewed about these requirements as well as the references to these concepts that were found in related literature [5][13].

Other requirements, particularly the security requirements, are deemed less generalizable than the other requirements, but still fairly generalizable to companies operating in regulated countries or regions. For example, the requirement of local storage primarily stems from regulatory considerations related to the origins of client data from different countries of the world. Therefore, this requirement may not be relevant to companies operating in a single

country or geographic location. Similarly, the requirement to remove personal data mainly arises from the sensitive nature of the data handled by the case company, which necessitates compliance with EU regulations such as GDPR laws. Companies located outside the EU may not have similar regulatory obligations.

Regarding the first design solution and its subsequent implementation, we conclude that the result, i.e. that the design solution is ultimately not satisfactory, is generalizable. The reason for this is that the solution fails to support some of the most generalizable requirements to a satisfactory degree. Additionally, the tool the solution is based on is not designed for the specific purpose of managing data in a machine learning context. This necessitates the management of certain data components in a separate system (git), contributing to the system's overall cumbersome usability.

It is possible that the second design solution could be generalized to any company who has the financial means to implement it. However, it still needs to be tested beforehand to verify its feasibility in validating the requirements as intended in the design solution.

6.4 Related Work

In this section, we will discuss and reflect upon related work pertinent to this thesis and the problem domain. This section aims to highlight the findings and differences observed in the work of others who have addressed similar problem. It is worth noting that we encountered difficulty in finding relevant research related to their problem domain, which also subsequently influenced the approach taken during the design phase.

6.4.1 Data Management Challenges for Deep Learning

In this paper[4], the authors conducted a multiple-case study where they investigated seven different organizations that were developing Deep Learning (DL) applications using real-world data for training. The data was collected through semi-structured interviews with DL experts from the different cases.

The motivation behind the study was that while there has been significant progress in the field of DL, the development of methods to manage the data needed for DL models has been lacking. The goal of the study was to identify and categorize challenges related to data management in development of deep learning models.

The study outlines 20 different data management challenges and categorizes them according to the development phase in which they are encountered. Some examples of challenges identified are shortage of diverse samples, data leaking (training and validation data getting mixed) and data shifts. Overall, the challenges identified in this paper are more related to the data itself, rather than the management of data which makes their results harder to compare to our results. According to the authors, the challenges identified in this study could be used by practitioners to foresee roadblocks that may occur while developing DL-applications, or as research challenges for the academic community.

The study concludes that because the management of data often requires more effort and time than the actual model creation, companies building DL applications that are able to

manage their data well and overcome these challenges can have a significant advantage.

6.4.2 DataHub: Collaborative Data Science & Dataset Version Management at Scale

In this paper[13], Bhardwaj et al. argues that current data management systems lack the capability to effectively support collaborative data environments, and no previous tool has explicitly addressed the problem of dataset versioning. For instance, relational databases, although suitable for schema-conformed data, do not offer any version management support. Consequently, teams often resort to storing data in file systems and rely on manual version management techniques, which is similar to the current situation encountered at the case company. Furthermore, the authors evaluated established source-code version control systems like SVN and Git and discovered significant performance issues. They contend that these systems are not designed to handle millions of files or terabytes of data.

To address the problem of dataset versioning at scale, the paper proposes a two-tiered system: a dataset version control system that enables developers to perform actions on such as branching and merging on large collections of datasets, and a platform (DataHub) which enables users to perform collaborative data analysis, leveraging the aforementioned version control system.

The authors conducted a survey among several computational biology groups at MIT to underscore the necessity of a dataset version control system. They argue that their results are representative of many other data science teams across various domains. The problem addressed in this paper, along with the results of the survey which served as the motivation behind this research, closely mirrors the challenges we aim to address at the case company where this thesis is conducted. Both focus on the challenge of data versioning at a large scale in a collaborative environment. However, there is a distinction in the primary use of the data. The teams interviewed in this paper primarily used data for data analysis and related tasks, whereas the case company employs the data for ML training purposes.

Despite being published in 2014, the paper highlights the absence of tools explicitly designed to address the issue of data versioning, a finding that aligns with our observations. The paper's contribution to future work is notable, as it presents a detailed design solution that can be implemented, evaluated and further developed.

6.4.3 Data Platform for Machine Learning

This paper[5] is published by a number of employees at Apple Inc. in 2019. Unfortunately, we did not find it in our initial research and were therefore unable to include learnings from this paper in our requirements or design. On the other hand, this means that we can compare the conclusions we have made with the ones made by the authors of this paper to see if we can find any similarities or significant differences.

The paper describes a new solution, called Machine Learning Data Platform (MLdp), which addresses the challenges identified in existing solutions. Some of the challenges identified is that data is currently often stored in multiple places for different parts of the development process or when handled by different teams, that the data needs to be tracked and versioned in order to ensure reproducible results, ensure compliance with regulations

and identify stakeholders, that there are performance challenges, and more. Towards the end of the paper the authors also include a call to action from their communities in order to solve further challenges, such as data discoverability and visualization, optimizations and integrations.

Along the way the authors describe some interesting and well thought out ways to solve some of the challenges, as an example, the authors come up with a way to divide the data into separate configuration items: 1. Dataset - the main entity on which the models are trained on, which is mostly immutable once it has reached a mature level. In a computer vision project, for example, the data sets could consist of images.. 2. Annotation - labels describing the data. 3. Split - a subset of a dataset, often used to divide the dataset into a training subset and validation subset. 4. Package - packages are similar to views in databases and are basically a selection of data from different datasets. One can think of them as a way to construct new datasets out of existing data. This division of the data has some advantages. As an example, new datasets can be created from existing data without affecting previous data sets or requiring any duplication of data, datasets can have multiple sets of annotations serving different purposes, and the dependency of models can be tracked accurately. The authors are using their own experiences in machine learning and databases, as well as related work when they identify and solve problems. It is implied in the paper that MLdp was built and is not just a theoretical system, but there is no information on if it is used anywhere or if it's available to the public. The authors suggest that MLdp will allow for better collaboration and innovation for organizations, but recognize that there are still challenges to address.

This paper raises several interesting questions. The authors describe a platform that addresses the most important challenges better than any other tool we have tested, but the platform appears to be unavailable to the public. Why has the platform not garnered any significant attention or adoption? Is it flawed in some non-obvious way or was it only ever meant as a proof of concept? The paper includes detailed technical descriptions, so why has nobody else picked up the torch? From this paper and related papers as well as from our experiences with the case company we can conclude that the problems discussed here are real, so possible reasons are that other organizations are using in-house solutions or are unaware of the extent of the problems.

6.4.4 Discussion

Upon comparing the results of this thesis with those from the reviewed literature, many similarities can be noted. Throughout the literature it is noted that although significant efforts have been put towards some ML problems, such as algorithm development and model management, there has been a lack of progress when it comes to management of the data needed to develop the models [4][5]. This was a conclusion that we also arrived at during the time we conducted our thesis work.

Regarding requirements and challenges, Wu et al. [5] identified several challenges as part of their work on a machine learning data platform. Similar to our findings, they have identified versioning, traceability and data lifecycle management as some of the major challenges. They also discuss concepts such as data visualization, data discovery and data exploration, which we find to be related the requirements of searchability and overview functionality we've elicited in this thesis. Additionally, they address performance optimizations, which correlate with the requirement of scalability that we've identified, as well as the ability to

completely remove data for regulatory compliance purposes. Furthermore, Bhardwaj et al.[13] also highlight that the system they proposed must be able to handle large datasets, which closely aligns with the scalability requirement elicited in this thesis.

Bhardwaj et al.[13] also noted challenges encountered by the teams they surveyed, including the use of a shared file folder for data storage, unknown data duplication, as well as the expressed desire for a transparent data access mechanism. These challenges closely resemble those encountered at the case company, aligning with the initial problem we identified after analyzing the current situation at the case company. Wu et al.[5] also discuss access control. This is something that we did not include in our requirements but still ended up working towards in the final design and implementation, and perhaps it should have been included in the original requirements.

Some of the requirements we have identified, such as local server storage and encryption of data, we deem to be quite specific to the case company, which may explain their absence in related literature. We also believe that some authors of the paper we've examined might have overlooked discussing the need to load only required data, perhaps assuming it's a bit of a no-brainer. Surprisingly, we found no mention of avoiding the shared data problem or the simultaneous update problem in the literature we have examined. This was unexpected, considering it is one of the main features of similar systems built to manage source code.

6.5 Future Work

In this section, we will discuss and suggest potential areas to be further researched in the future. By doing so, interested readers may gain insights and inspiration for further exploring the subject matter. Because of the time limitations of this thesis project, we couldn't carry out the implementation and evaluation of the second design solution based on Helix Core. This area could be further explored as future work for those aiming to address a similar problem as the one at hand in this thesis. Furthermore, since the proof of concept wasn't evaluated by developers, this aspect could further be investigated.

One of the conclusions drawn from this thesis is that a comprehensive purpose built solution meeting all the requirements likely does not exist. For this reason, an interesting future work would be to develop and implement such a solution. We believe that with the requirements elicited and outlined in this thesis and mentioned in related work, along with the technical solutions discussed and presented in the work of Wu et al.[5] and Bhardwaj et al.[13], collectively demonstrate substantial potential for the creation and a new industry-defining system.

Chapter 7
Conclusion

The purpose of this thesis was to investigate if there is a scalable solution for storing and managing training data. To achieve this we conducted an investigation into the current process of handling the data at the case company, followed by interviews with developers and a literature study. The result was a comprehensive requirement specification. The next step was to develop design solutions based on the requirement specification, which resulted in two design solutions. One of the two design solutions was implemented in a proof of concept. Reflecting on the main research question, **RQ1:** *How can a training data storage repository be designed to improve data management processes?* This thesis has outlined several aspects that can enhance the data management process. The most important and generalizable requirements are versioning, scalability, traceability and data lifecycle management. Additionally, the requirements of overview functionality, avoiding the shared data/simultaneous update problem, automatic validation and searchability are also considered important requirements.

The first design solution is an idealized solution based on the possibilities of binary repository managers. Although the solution wasn't tested during the design phase, we believed it would fulfill 7 of the 12 requirements to a satisfactory degree, and 3 of the remaining requirements would be partially fulfilled. Design solution 2 was based on Helix Core, but was deemed financially unfeasible for the case company to implement. Therefore, and due to time constraints of this thesis project, it was not implemented in a proof of concept.

During the development of a proof of concept for design solution 1, it became evident that the tool was not intended for the management of training data used in machine learning development. While the solution did meet some requirements, many requirements were not met to the expected degree, leading to the realization that it was ultimately insufficient as a complete solution to the problem. Based on these findings and related literature that we have reviewed, we conclude that a publicly available tool capable of meeting the requirements outlined in this thesis likely does not exist. Therefore, we propose future endeavors to develop such a solution, implementing the requirements outlined and discussed in this thesis along with the technical solutions presented in the related literature[5][13].

References

[1] Kristina Säfsten and Maria Gustavsson. *Research Methodology For Engineers and Other Problem-Solvers*. Studentlitteratur AB, 2020.

[2] Wayne A Babich. *Software configuration management: coordination for team productivity*. Addison-Wesley Longman Publishing Co., Inc., 1986.

[3] Amazon. What is SDLC (Software Development Lifecycle)? `https://aws.amazon.com/what-is/sdlc/`. Accessed: 2024-03-27.

[4] Aiswarya Munappy, Jan Bosch, Helena Holmström Olsson, Anders Arpteg, and Björn Brinne. Data Management Challenges for Deep Learning. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 140–147, 2019.

[5] Pulkit Agrawal, Rajat Arya, Aanchal Bindal, Sandeep Bhatia, Anupriya Gagneja, Joseph Godlewski, Yucheng Low, Timothy Muss, Mudit Manu Paliwal, Sethu Raman, Vishrut Shah, Bochao Shen, Laura Sugden, Kaiyu Zhao, and Ming-Chuan Wu. Data Platform for Machine Learning. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 1803–1816, New York, NY, USA, 2019. Association for Computing Machinery.

[6] Scandinavian Network of Excellence in Software Configuration Management. `https://snescm.org/Common/CMCM/`. Accessed: 2023-08–15.

[7] Alectio. `https://alectio.com/`. Accessed: 2023-11-02.

[8] Aquarium. `https://www.aquariumlearning.com/`. Accessed: 2023-11-02.

[9] JFrog. Atifactory. `https://jfrog.com/artifactory/`. Accessed: 2023-11-02.

[10] DVC.AI. `https://dvc.org/`. Accessed: 2023-11-02.

[11] Eiffel. `https://eiffel-community.github.io/`. Accessed: 2023-11-02.

[12] Git Large File Storage (LFS). `https://git-lfs.com/`. Accessed: 2023-11-02.

[13] Anant P. Bhardwaj, Souvik Bhattacherjee, Amit Chavan, Amol Deshpande, Aaron J. Elmore, Samuel Madden, and Aditya G. Parameswaran. Datahub: Collaborative Data Science & Dataset Version Management at Scale. *ArXiv*, abs/1409.0798, 2014.

[14] Perforce Helix Core. `https://www.perforce.com/products/helix-core`. Accessed: 2023-11-02.

[15] Hydra. `https://hydra.cc/`. Accessed: 2023-11-02.

[16] Labelbox. `https://labelbox.com/`. Accessed: 2023-11-02.

[17] LatticeFlow. `https://latticeflow.ai/`. Accessed: 2023-11-02.

[18] Openlayer. `https://www.openlayer.com/`. Accessed: 2023-11-02.

[19] Sonatype Nexus. `https://www.sonatype.com/products/sonatype-nexus-repository/`. Accessed: 2023-11-02.

[20] Perforce. Helix Server As a Version Control Implementation. `https://www.perforce.com/manuals/overview/Content/Overview/basic_concepts.helix.html`. Accessed: 2023-11-02.

[21] Perforce. Install Helix Server on Linux. `https://help.perforce.com/helix-core/quickstart/Content/quickstart/admin-install-linux.html`. Accessed: 2023-11-02.

[22] Perforce. Helix Visual Client (P4V) for Helix Core. `https://www.perforce.com/products/helix-core-apps/helix-visual-client-p4v`. Accessed: 2023-11-02.

[23] Perforce. Helix Core. `https://www.perforce.com/products/helix-core`. Accessed: 2023-11-02.

[24] Perforce. The Basics of Version Control. `https://www.perforce.com/manuals/overview/Content/Overview/basic_concepts.basics.html`. Accessed: 2023-11-02.

[25] Perforce. Performance, Scaling, and High Availability. `https://www.perforce.com/manuals/overview/Content/Overview/basic_concepts.performance.html`. Accessed: 2023-11-02.

[26] Perforce. Securing the System. `https://www.perforce.com/manuals/overview/Content/Overview/basic_concepts.security.html`. Accessed: 2023-11-02.

[27] Reeve, April. Chapter 22 - Conclusion to Managing Data in Motion. In Reeve, April, editor, *Managing Data in Motion*, MK Series on Business Intelligence, pages 160–163. Morgan Kaufmann, Boston, 2013.

[28] Loshin, David. Chapter 10 - Data Consolidation and Integration. In Loshin, David, editor, *Master Data Management*, The MK/OMG Press, pages 177–180. Morgan Kaufmann, Boston, 2009.

[29] JFrog. JFrog Installation & Setup Documentation. https://jfrog.com/help/r/jfrog-installation-setup-documentation/install-artifactory-single-node-with-debian. Accessed: 2023-11-02.

[30] JFrog. JFrog Platform Administration Documentation, Environments. https://jfrog.com/help/r/jfrog-installation-setup-documentation/install-artifactory-single-node-with-debian. Accessed: 2023-11-02.

Appendices

Appendix A
Interview Questions

A.1 Pre-analysis Interview Questions

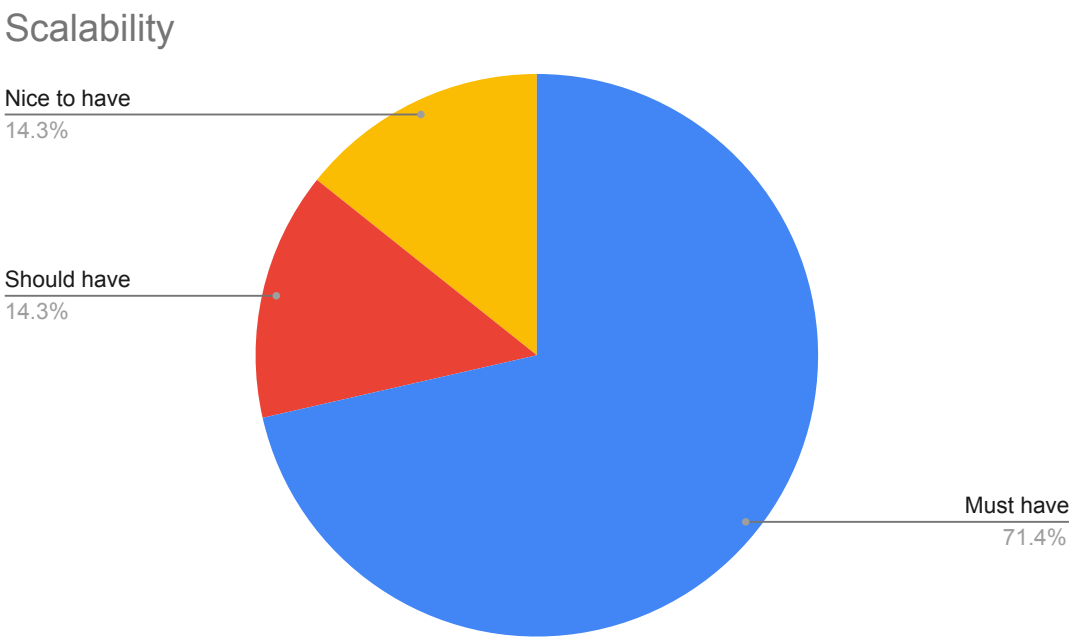
1. How would you describe the project you are working on to someone outside the organisation?
2. What type of development are you currently working on?
3. What product do you deliver?
4. What tools do you use to keep track of project requirements?
5. What do you think is the most difficult part of your job? What are the biggest challenges?
6. Why do you think these challenges occur?
7. How do you approach a problem that occurs? Which steps do you take from the initial stage to the final stage on a specific problem you're working on?
8. What do you think is the easiest part of of your job?
9. How do you handle the current different variants within the project?
10. What works well with having and handling many variants, and what doesn't work well?
11. Describe a typical scenario during your workday when you interact with FPDB?
12. How long does it take you to find the right data?
13. How is what the customer wants represented to you?
14. If you were to describe FPDB to a new employee, how would you do it?
15. How would you describe the index files and their purpose?

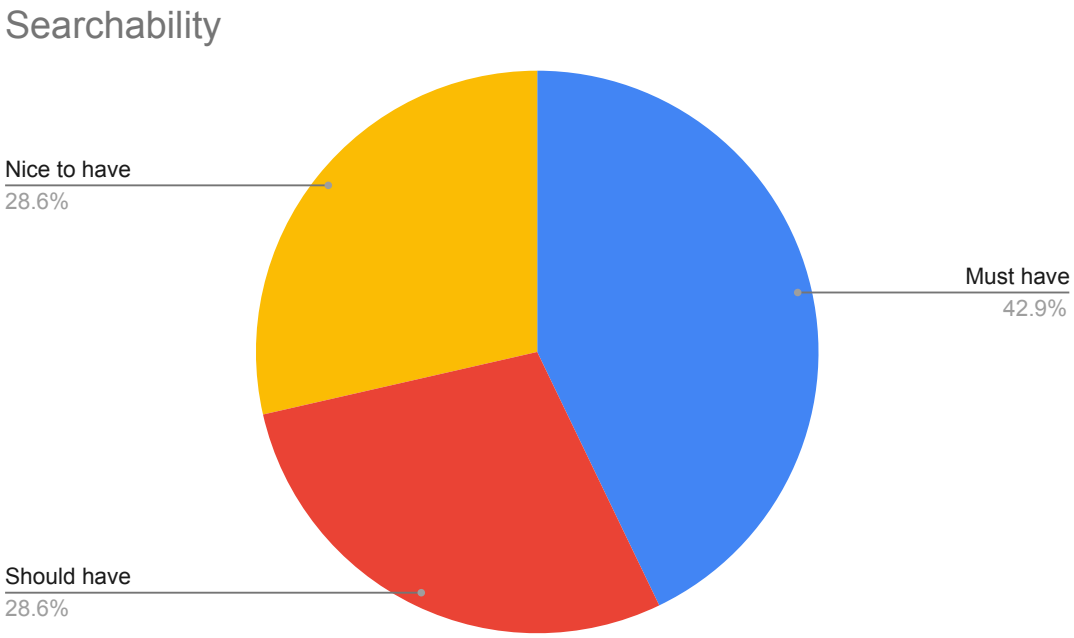
A.2 Requirement Elicitation Interview Questions

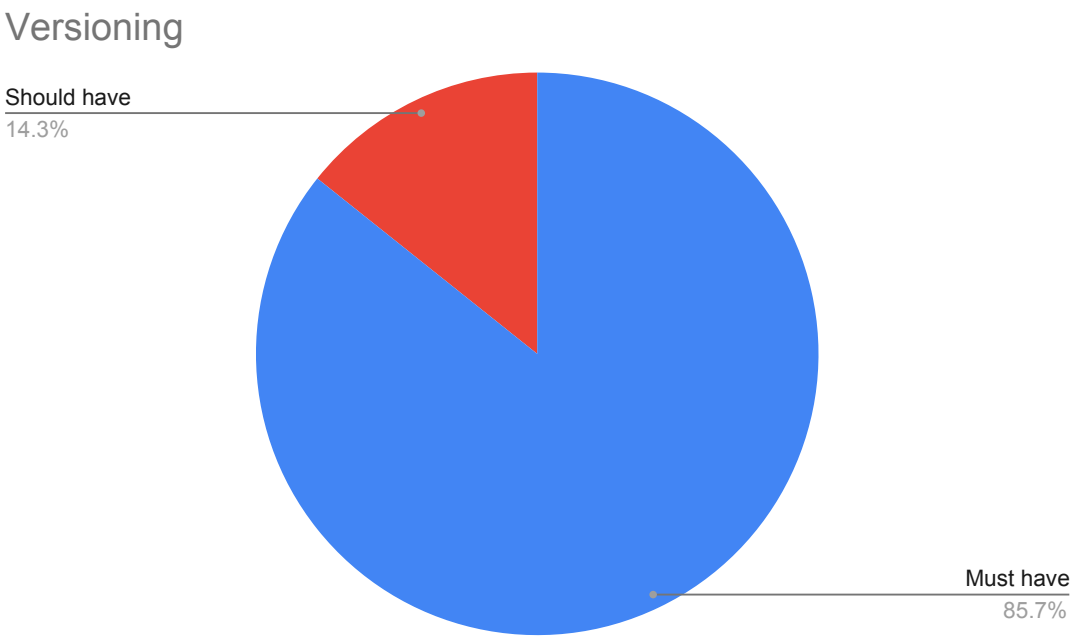
1. Are there any problems with the way you work that you think should be addressed?
2. Which functionality is desired but missing in FPDB (generally)?
3. Is there any specific functionality that is missing but would aid you personally in your workflow?
4. How long does it take you to use FPDB?
5. Can you describe in detail the whole process of handling the data, from when you receive it from the customer until it is used in training the model?
6. How do you see the difference between processed and non-processed data?
7. How does the system store processed data and non-processed data?
8. What are the inputs and outputs for the whole system (training data, metadata, what else?)?
9. What metadata is required (for each image/data)?
10. Rank the following requirements from must have - should have - nice to have or not needed.

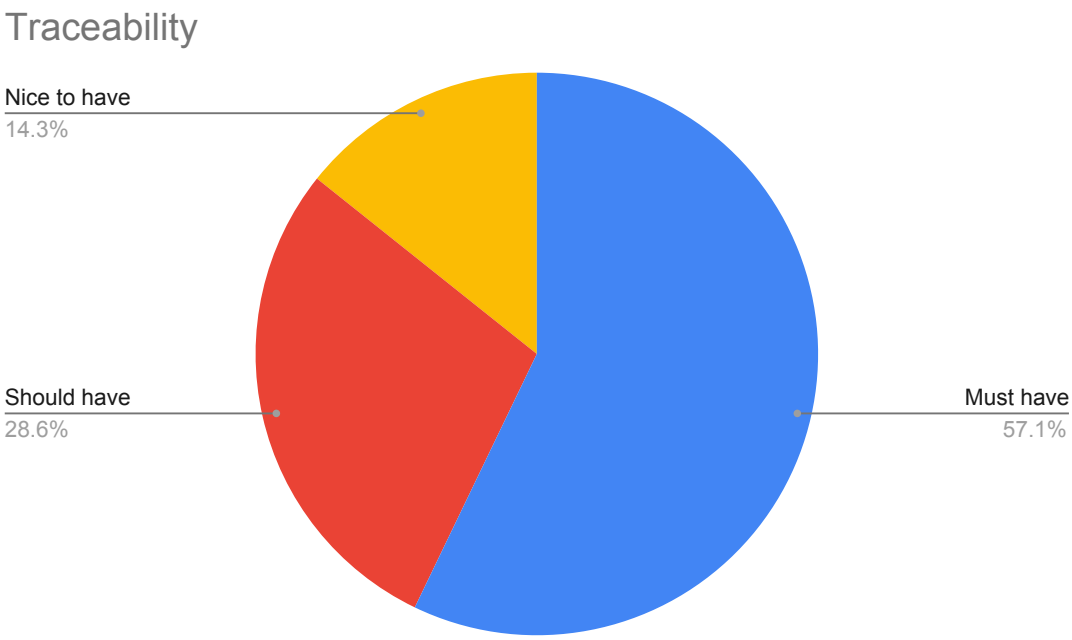
Appendix B

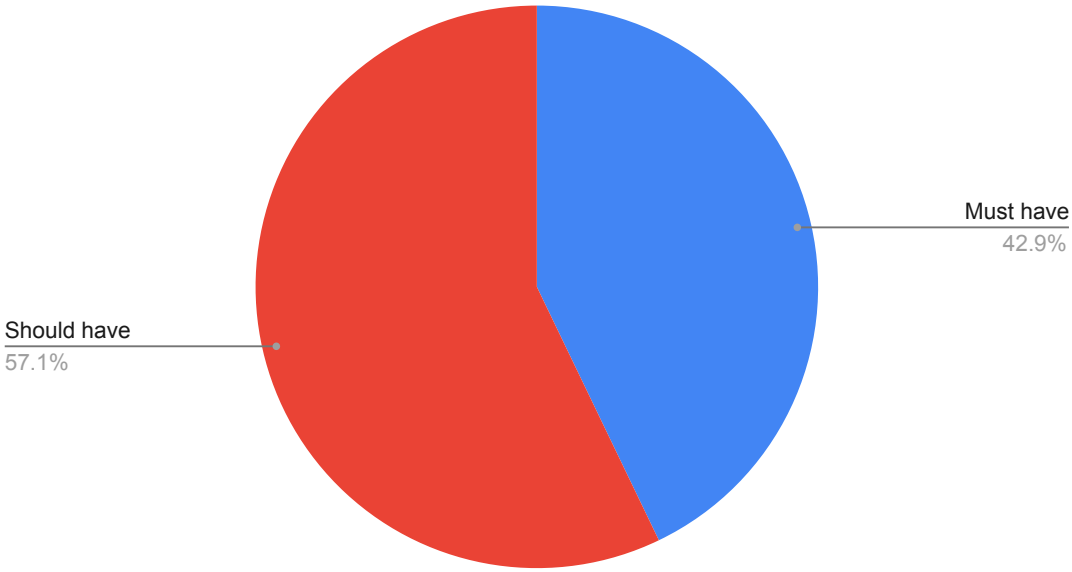
Ranking of Requirements by Interviewees

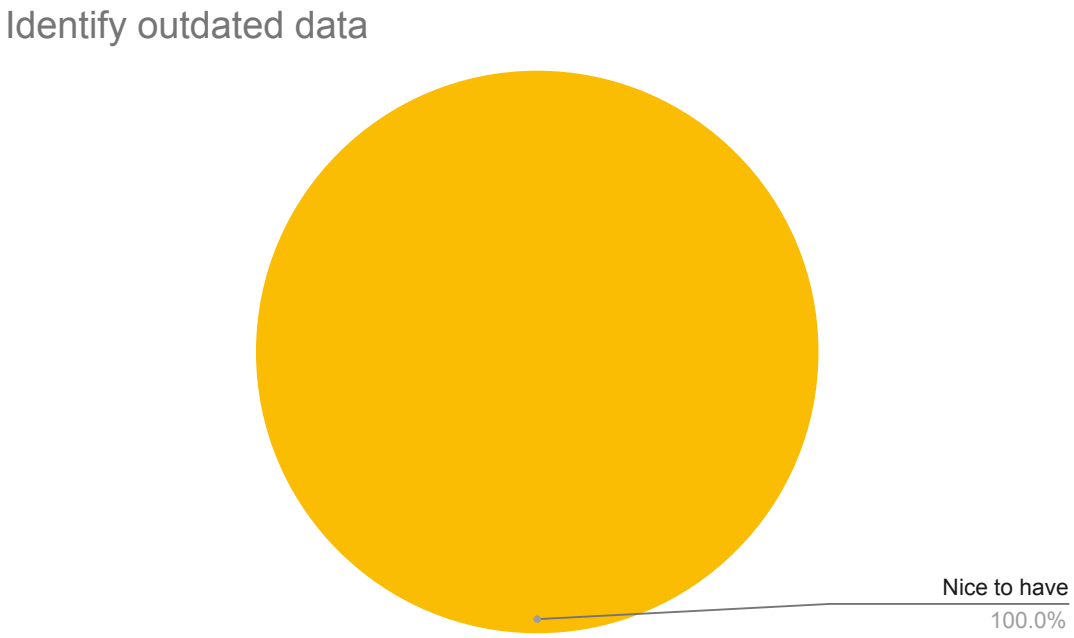


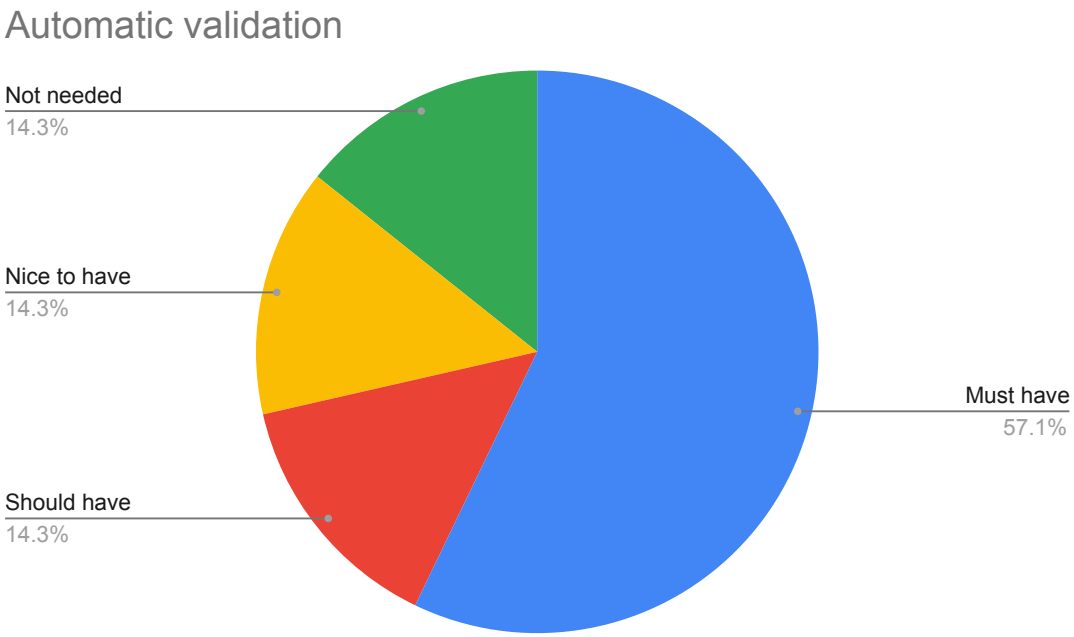

Searchability
Nice to have
28.6%
Must have
42.9%
Should have
28.6%

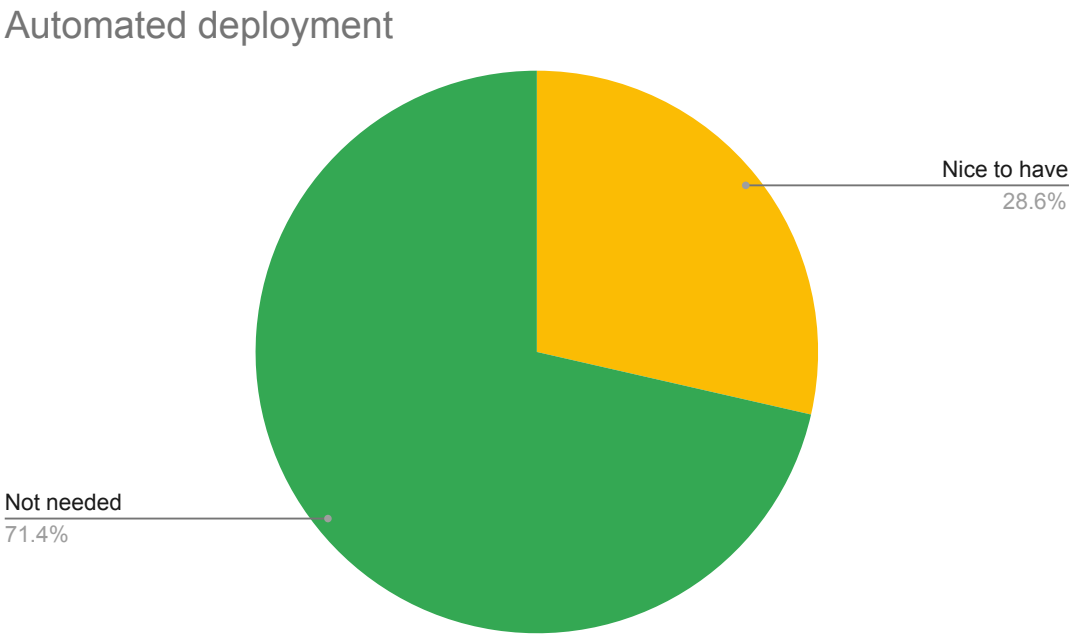

Versioning
Should have
14.3%
Must have
85.7%

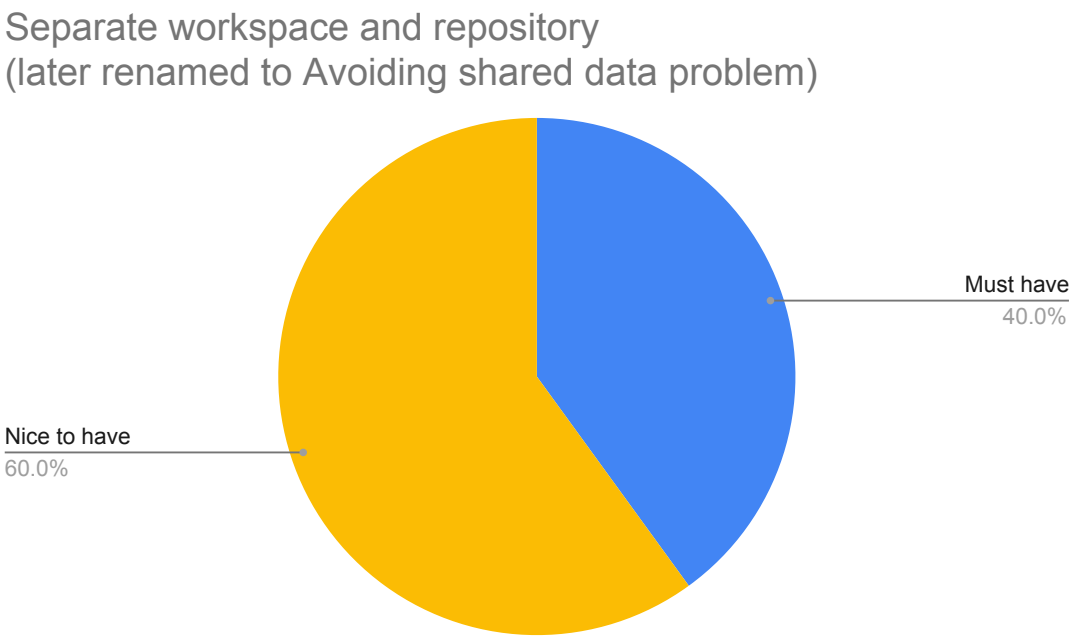

Traceability
Nice to have
14.3%
Should have
28.6%
Must have
57.1%

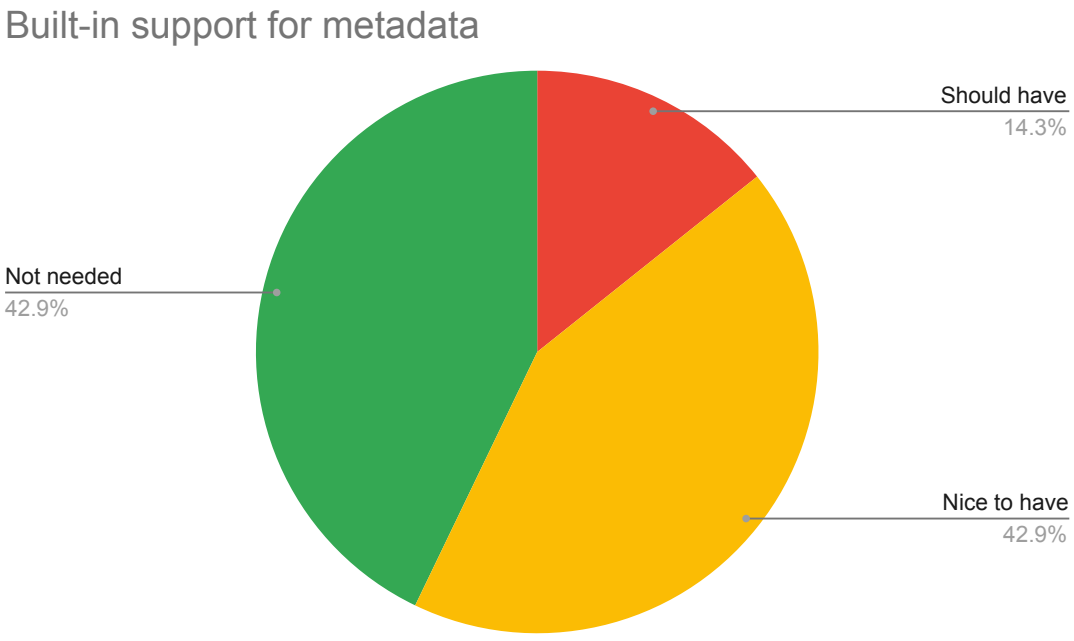

Staging (later renamed to Data Lifecycle Management)
Must have
42.9%
Should have
57.1%

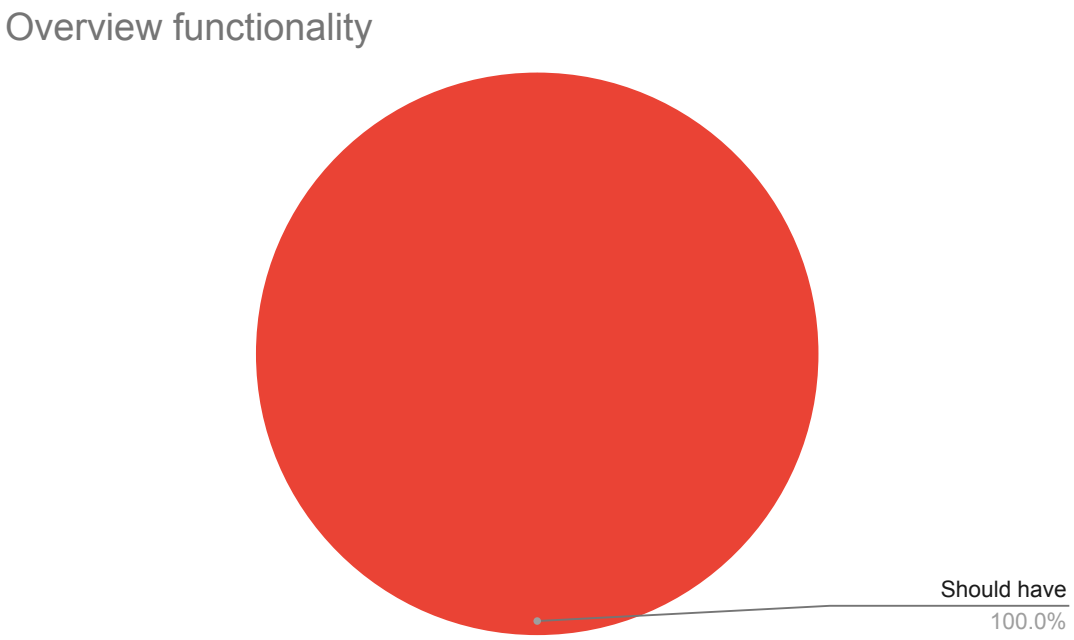

Identify outdated data
Nice to have
100.0%


Automatic validation
Not needed
14.3%
Nice to have
14.3%
Should have
14.3%
Must have
57.1%


Automated deployment
Nice to have
28.6%
Not needed
71.4%


Separate workspace and repository
(later renamed to Avoiding shared data problem)
Must have
40.0%
Nice to have
60.0%


Built-in support for metadata
Should have
14.3%
Not needed
42.9%
Nice to have
42.9%


Overview functionality
Should have
100.0%

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2024-03-14

EXAMENSARBETE Management of Training Data for Deep Learning Applications: Requirements and Solutions
STUDENTER Adla Lagström Jebara, Fabian Sundholm
HANDLEDARE Lars Bendix (LTH)
EXAMINATOR Emelie Engström (LTH)

Uncharted Territories: Exploring Data Management in Software Development

POPULÄRVETENSKAPLIG SAMMANFATTNING **Adla Lagström Jebara, Fabian Sundholm**

Within software development, the management of source code has been extensively studied. The same level of attention, however, has not been devoted to the management of associated data, such as training data used in machine learning algorithms.

In recent years, deep learning technologies have advanced rapidly, which has resulted in the launch of several new applications such as image generating models and chatbots. These applications are trained on large datasets to recognize patterns and make predictions or classifications based on new data they encounter. However, despite deep learning being almost a household name at this point, very little is known about how the companies developing these applications manage the vast amount of data that is required to create these applications. There is also very little public research available on data management in a machine learning context.

At the case company where this research was conducted, machine learning algorithms are used in combination with other technologies, such as conventional image analysis methods, to provide tailored fingerprint recognition solutions, i.e. technology that identifies and verifies individuals based on their unique fingerprint patterns. However, the process of working with the vast amount of data have been described by developers at the company as "frustrating", "complex", and "quite messy", and suffers from numerous challenges.

To address this problem, our research aimed to investigate if there is a scalable solution for storing and managing training data, and thereby enhance the effectiveness of the developers.

The research included identifying the system's needs and how it should work, developing designs to meet these needs, and then testing out one of these designs to see how it works.

To identify the system's needs, we did a literature study and interviewed several developers at the case company. To come up with design solutions, we looked at available tools to determine if any existing tool could fulfill the system's needs. We found two different design solutions, and one of them was selected for implementation as a proof of concept.

However, we found that the proposed design solution did not fully meet the system's needs, indicating a complexity in addressing the problem beyond our initial expectations. Moreover, we found that an available tool on the market fulfilling all the system's needs to a satisfactory degree likely does not exist. Nevertheless, our research indicates that with additional time and resources, it is feasible to address the problem and develop such a solution by implementing the identified needs of the system, which we be consider to be an interesting area for future work.