MASTER'S THESIS 2026

Creating an Onboarding Tool for Technical Integration Using an LLM Approach

Axel Nåsell, Harry Norrman

ISSN 1650-2884
LU-CS-EX: 2026-15

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2026-15

Creating an Onboarding Tool for Technical Integration Using an LLM Approach

Ett LLM-baserat introduktionsverktyg för den tekniska integrationsprocessen

Axel Nåsell, Harry Norrman

Creating an Onboarding Tool for Technical Integration Using an LLM Approach

Axel Näsell
ax1857na-s@student.lu.se

Harry Norrman
ha5315no-s@student.lu.se

May 18, 2026

Master's thesis work carried out at Robert Bosch AB.

Supervisors: Daniel Jung, Daniel.Jung2@se.bosch.com
Sule Tekkesinoglu, sule.tekkesinoglu@cs.lth.se

Examiner: Elizabeth Bjarnason, elizabeth.bjarnason@cs.lth.se

Abstract

Technical integration of new developers in the onboarding process is an important aspect for companies to grow their workforce efficiently. This is a difficult process that often leaves new developers confused and demands resources from experienced developers to get them started. This thesis explores the possibility of creating an LLM-based assistant that can expedite the onboarding process. Through the design science research methodology, we identified current issues related to technical integration within the onboarding process, implemented an agentic AI solution called Bosch Onboarding Buddy (Bob) and evaluated its performance through user studies. In relation to current challenges, we found that the biggest issue is finding the most relevant information related to a task assignment within a large documentation database. We implemented Bob using Retrieval-Augmented Generation (RAG) to enable real-time information fetching to solve this problem. Our evaluations showed promise in the ability to find and summarize relevant information. While Bob cannot replace the value of receiving assistance from senior developers, it complements the existing technical integration tools and strategies.

Keywords: AI Tooling, RAG, Onboarding, Agentic, Software development, LLM

Acknowledgements

We would like to thank Bosch and the employees in our team for providing us with the environment and tools needed for completing this thesis. Their willingness to provide access to project documentation as well as participating in tests has been vital for the success of our project.

Special thanks to our supervisor Sule Tekkesinoglu for her guidance and feedback throughout the entire process. Her swift responses and regular supervision of the project made sure we stayed on track.

We would also like to thank our examiner Elizabeth Bjarnasson who supported both us and Sule with her experience in conducing a master thesis.

Contents

Chapter 1
Introduction

The onboarding process in software development is a time-consuming task that involves many different steps that collectively hinder new developers from creating meaningful contributions to a software project. One of the more prominent steps in the onboarding process is the technical integration, where developers need to familiarize themselves with the codebase, frameworks and practices used in a specific project.

With the breakthrough of generative artificial intelligence (Gen AI), specifically large language models (LLMs) in recent years, many companies search for new solutions to old problems that have previously been difficult to overcome. We propose that an AI-based assistant can be employed to reduce the need for continuous support from senior developers during the onboarding process. Reducing arduous reading for new developers by creating tailor-made responses that explain any given aspect of a project.

Currently LLMs show great potential but with some common pitfalls. One of the most discussed drawbacks is the hallucination, i.e., generation of false or misleading information. Instruction adherence is another common pitfall, referring to how well the LLM is able to follow the instructions that are given by the user in combination with the context that it is used. This becomes more difficult as the prompt complexity increases, as well as when the context given to the LLM becomes too large. When this happens responses can deteriorate in quality and focus. Retrieval-Augmented Generation (RAG) is a technique to help alleviate problems with hallucination and instruction adherence, providing relevant context to a given prompt.

Another important part of creating any development tool is to consider the operating expense of using such tools in everyday work. Different LLMs generate responses with varying speed, accuracy and cost per token, and it is important to identify which LLM is most efficient for the purpose that it is intended to serve.

This project has been conducted at Bosch R&D center Lund, where we have been given access to developers and resources to create and evaluate this system. Bosch is a multinational tech-company with a very diversified business that ranges from home appliances to sensor technology. A large part of this includes software development towards many dif-

ferent products. The aim of this project is to create an LLM-based system, called Bosch onboarding buddy (Bob) that speeds up the technical integration step of the onboarding process, assisting new developers with their introductory work. Technical integration can however be very different depending on what products and processes are in place at a company. Therefore, the current problem areas need to be explored and identified thoroughly. A solution is proposed using a combination of modern techniques, specifically RAG, agents and prompt engineering. Three research questions guided this process:

RQ1 What are the current challenges in the technical integration process?

RQ2 How can an LLM-based onboarding system be designed to support the identified needs of the current technical integration process?

RQ3 How does the LLM-based onboarding system support the technical integration process?

By following these research questions, we first gained insight into the current technical integration process discussed in the literature, as well as the problems faced at Bosch. Then we leveraged AI to develop an onboarding tool to address these problems and evaluated the effectiveness of the proposed system. The outcomes of this thesis contribute to the research in AI tool development as well as provide the Bosch with insights into their technical integration process and an AI-based onboarding tool to assist in technical integration.

In the rest of this report, we cover the following: Background information that is needed to understand the project (Chapter 2), the research methodology that we follow to conduct and evaluate our research (Chapter 3), the evaluation results of the proposed solution (Chapter 4), analysis and discussion of the results (Chapter 5), and finally the conclusions of the thesis (Chapter 6).

Chapter 2
Background

2.1 Onboarding Process

The onboarding process refers to the integration of an employee into a workplace or organization [9]. It often includes several parts such as ways to socialize, company culture, getting familiar with projects and steps into succeeding at your work. An onboarding can look different depending on the work as well as the company setting. The onboarding at a storage facility versus an office would look very different. For instance, the storage facility may focus more on workplace safety as risks are higher while an office may have a larger focus on navigating the office space. This work focuses on the onboarding of software developers, specifically what is called technical integration, which is discussed in Section 2.1.1.

2.1.1 Technical Integration

The technical integration is the part of the onboarding process where tools, codebase, frameworks and code practices are learnt and presented. The term 'technical integration' is not usually used to describe this part of the onboarding, however for the purposes of this thesis, there is a need to differentiate between this and other onboarding activities.

There exists some strategies that can be useful to improve the technical integration process. The first thing that one needs to consider is task assignment, which refers to which type of tasks a new developer is given when familiarizing themselves with a new project [20], which is covered in Section 2.1.2.

There are also a wide range of onboarding techniques which are smaller activities that help developers with the technical integration process. These techniques can be divided into three categories: activity, people, and artefact [11]. Examples include mentoring, peer support, online information searching, reviewing code. We have grouped these techniques by context and discussed them in Section 2.1.2, Section 2.1.3, and Section 2.1.4.

2.1.2 Task Assignment Strategies

In software development there are mainly three types of task assignment strategies when onboarding: simple-complex, priority-first and exploration-based [20]. **Simple-complex** as the name suggests assigns easy tasks and gets progressively harder. Simple tasks often involve handling isolated components with less impact to a system and is meant to give a developer more confidence when working with a system. **Priority-first** on the other hand assigns important tasks first. The difficulty of the tasks can vary but is generally more difficult than simple-complex uses. This task assignment strategy is meant to push a developer into an actual workflow fast, making them generally more comfortable in high-stress situations. Lastly, **exploration-based** assignment does not give a specific task to a developer, but instead allows the developer to choose what to work on, making them find their special interest in a project faster.

2.1.3 In-Person Support

In-person support refers to methods that involve more people than the onboarder [11]. Common strategies include mentoring, peer support, pair programming and team meetings. In person strategies are widely used in software development as work often involves very large or complex systems, which can be daunting to tackle as a new developer. Therefore in person support is used to give onboarders someone to lean on and ease them into the system. **Mentoring** is a commonly used strategy and assigns a mentor to an onboarder as a specific person to ask questions. The mentor should be familiar with the system the onboarder works on and can help with answering questions that might arise. **Peer support** is the more general version of mentoring. It is not assigning a specific person, but making the entire team a wealth of knowledge, allowing and encouraging asking questions to the most relevant person. **Pair programming** is a more niche practice but is the act of programming in a pair allowing for fast feedback and very little friction in question asking and answering. Lastly, **team meetings** provide general support for the technical integration, which naturally happens in most workplaces. It is not for specific information but more general guidance or a start to other strategies such as peer support.

2.1.4 Self-Learning

Self learning is the most widely used method of learning in software development, it includes every method an onboarder uses to learn without the help of others. Self learning includes documentation reading, code reading and online searching. It is the most widely used method since it is almost completely integrated into the workflow of a software developer [20]. **Documentation and code reading** are almost mandatory in the case of any existing system. As the name suggests, it involves examination of the documentation and source code to understand the system structure and behavior. **Online search** is the last method, which involves searching for information related to the issues encountered using external resources, such as search engines and forums.

2.2 Generative AI

Generative AI has for the last couple of years become more and more popular. With the introduction of ChatGPT to the public, massive investments have flooded into the tech space for more research into better models [17]. Generative AI, specifically large language models and diffusion-based models are the big leaders in this space. Large language models are sequential text-generation models trained on large amounts of data [14]. Another class of generative AI gaining a large amount traction has been diffusion based models, which create content, often images out of random noise [14].

One of the largest changes in the AI space has been the introduction of transformer models presented in [31]. Transformer models are the foundation that many large language models use to increase performance during training, leveraging the use of the "attention" mechanism, making the task more parallelizable and therefore more easily scalable.

Since the introduction of large transformer-based models, multiple techniques have been introduced to improve their performance. Retrieval-augmented generation (RAG) and fine-tuning have been prominent techniques for quite a while [10]. New techniques are however being created all the time, one major example being Model Context Protocol, allowing for an API-like protocol to exist in the LLM space. When developing AI tools it is also very common to include the use of prompt engineering and different development packages, to further improve the performance of the system. In the following subsections, we describe these key components of generative AI in detail.

2.2.1 Retrieval-Augmented Generation

Retrieval augmented generation (RAG) is a method to increase performance in LLMs, providing context to LLM prompts by using external storage to retrieve relevant information [23]. Figure 2.1 illustrates a simple RAG-based system. Given a user query, a RAG application creates search parameters that are used to fetch documents from a data store. These documents are then provided alongside the user query to the LLM, which uses the documents as context to generate a response to the query. RAG can increase performance to LLM prompts by introducing context on data that a specific LLM model has not been trained on, thus lessening hallucinations [16]. RAG is therefore an important tool in the context of proprietary code or less public repositories, as it can be leveraged without the large upfront cost of training a model on their own. One common type of data-store is a vector database, which vectorizes data for efficient querying. One way to gather data for a RAG applications is through external databases, such as Confluence [7].

Vector Database

Vector databases are a storage solution for vectorized data, creating a resource-efficient way to store and index many kinds of data such as text, images and video [30]. A vector database works by storing n-dimensional vectors instead of relational data like a normal database would. Each entry in the vector represents some feature or part of a feature. These feature can be simple, such as numbers, or more complex such as color and texture in an image.

It is useful to create vectorized data because it allows for approximate similarity searches. By measuring the distance between vectors, it is possible to find the k closest vectors to a given

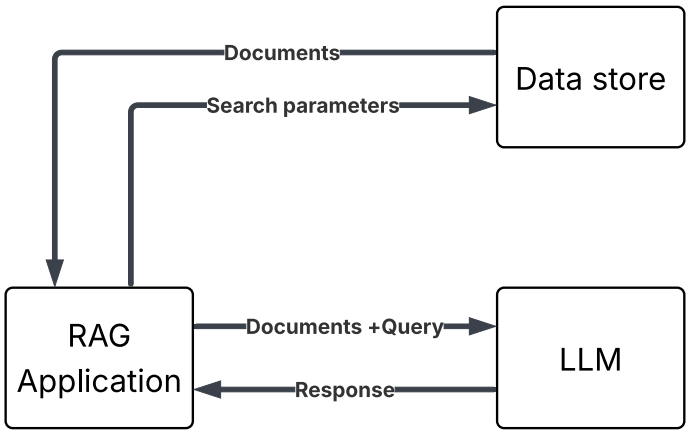


Figure 2.1: RAG-based system

query vector. This is a key component of RAG since this allows for fast information retrieval based on semantic search queries. The relevant data is prepared for RAG by vectorizing it using an *embedding model* that has been pretrained. Queries are vectorized using the same embedding model. Then a similarity search is conducted to retrieve the closest vectors which can be added as context to the LLM. A common strategy when preparing text data is to split the text into chunks. This makes sure that vectors represent substantially large blocks of text while also allowing for some overlap between chunks which further increases the accuracy of similarity searches.

Confluence

Confluence is an internal documentation database developed by Atlassian. It serves as the core documentation server for Bosch, called Docupedia. The database uses its own query language, namely the *confluence query language*, which has similarities to *SQL* but is less documented and somewhat less consistent in querying standard [7].

2.2.2 Model Context Protocol

Model context protocol (MCP) was introduced in late 2024 by Anthropic and provides a standardized protocol for LLMs to interact with tools, resources and prompts [6]. MCP works by a client-server architecture, where a client is hosted at an LLM-application and a separate server is established to allow access to user-decided primitives, as depicted in Figure 2.2. Primitives, including tools, resources and prompts, serve distinct functions. *Tools* can perform actions such as function calls, *resources* provide additional context such as file contents, and *prompts* are reusable templates for structuring answers and interactions.

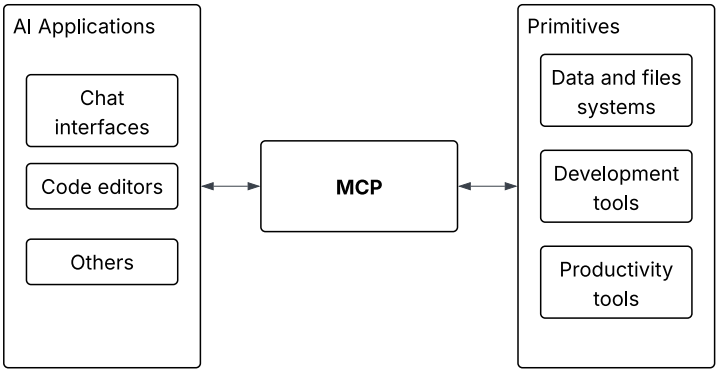


Figure 2.2: MCP Architecture

2.2.3 Prompt Engineering

Prompt engineering is an LLM technique that focuses on creating ideal prompts for an LLM in order to generate a desired response [32]. In essence, it is the way in which an LLM can be programmed through the use of prompts. By applying certain strategies and patterns to a prompt, we can alter the way an LLM generates its response. These strategies focus on the following: input semantics, output customization, error identification, interaction, prompt improvement and context control. *Input semantics* focuses on how an LLM can understand and translate user input and use that understanding to generate a response. *Output customization* focuses on structuring the answer from an LLM in the form of types, formats and structures. An example of this could be to tell the LLM that you want a response no longer than two sentences. *Error identification* makes sure that the LLM can reflect on its own response and it is factually correct, and if it is not, then the LLM revises its response. *Prompt improvement* uses the LLM to generate improved or alternative prompts that the user can use to get better responses. *Interaction* category focuses on changing the way the user interacts with the LLM. For example, LLM asks questions to the user until it has enough information to perform a given task that was stated in the beginning. Lastly, *context control* focuses on what context the LLM has and also what context it should consider when answering specific questions.

When designing an LLM-based system that uses internal LLM calls to construct specific chains of thought, it is important to consider how those internal prompts are engineered to make sure that the LLM responses are accurate and consistent.

2.2.4 Development Packages

When developing generative AI workflows it is common to use some combination of development packages. These include packages for orchestration of LLM applications, vector storage and retrieval, embedding models and agentic systems. In the following subsections, we present the packages that we have used for our project in detail, including LangChain, ChromaDB, Pocketflow, and embedding models.

LangChain

LangChain provides an open source framework with tools for developing LLM systems [4]. By using these tools, the process of implementing LLM-based systems becomes significantly more streamlined, since developers do not have to implement their own tools and data structures that are needed to use an LLM. The core function of LangChain is the seamless implementation of LLM agents. LangChain provides a framework in which an agent is easily created with a given LLM model and tools that are custom or provided via LangChain. It also has other useful features, such as the document data structure, which standardizes how a document should be structured to help passing data to and from LLMs. LangChain also has functionality to load data from external sources, such as git repositories, which can be used alongside text splitters to chunk the data for efficient storage in a database.

ChromaDB

ChromaDB is a vector database that provides storage of vector embeddings alongside metadata, vector search, text search, document storage, and metadata filtering. It can run locally as a server with python and typescript SDKs, focusing on ease of use, especially when creating prototype applications. ChromaDB is composed of six core components: gateways, distributed logs, query nodes, compactor nodes, system database and the vector storage (see Figure Figure 2.3). The *gateways* exposes an API and handles authentication. The *distributed log* records all writes before acknowledgments to clients to ensure atomicity across multi-record writes. The *query nodes* are responsible for read operations and enables vector similarity, text and metadata search. The *compactor nodes* periodically builds and maintains indexes by reading from the log and building updated vector, text and metadata indexes. The *system database* tracks tenants, collections and their metadata by using a SQL database [3]. Finally the *storage* represents the actual storage of vectorized data.

Pocketflow

Pocketflow is a python package that abstracts code into graphs [5]. Creating an easy way to modularize code by using a shared data storage and defining *flows* which guide nodes through a graph, as illustrated in Figure 2.4. It can be used to create multi-agent workflows, RAG systems and to implement other LLM-based strategies. The package has an open structure, where nodes are created for specific purposes and the flows determine in what order the nodes are executed, creating a graph structure.

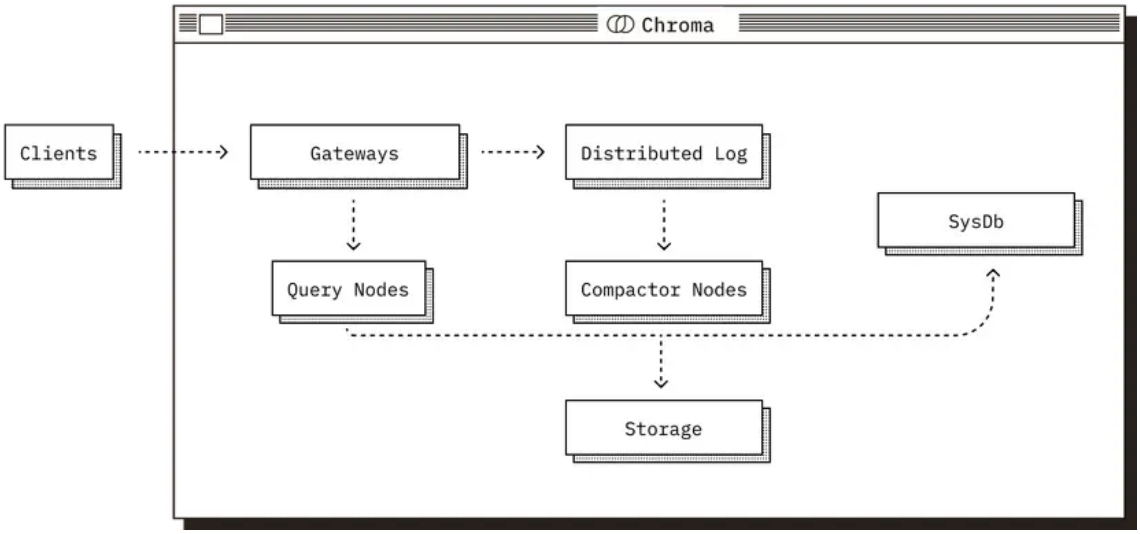


Figure 2.3: ChromaDB basic architecture

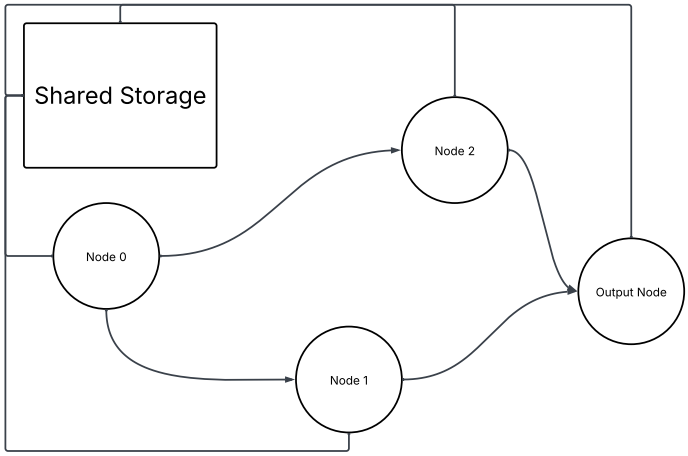


Figure 2.4: Pocketflow Graph Architecture

Embedding Models

There are various vector embedding models that are used in RAG systems. The implementation and vector sizes of these models make them perform with varying efficiency. Therefore, it is important to choose the right model specific to a given use case. For simpler scenarios, choosing a smaller embedding model can be the most practical choice, since larger models that perform better come at the cost of storage and processing power. Often these larger models can be accessed through the cloud by paying for the service in the form of tokens. Some smaller models can be run locally. While they do not have the same level of performance, they work sufficiently well for smaller projects. We have experimented using the following models: jina-embeddings-v2-base-en, all-mpnet-base-v2 and all-MiniLM-L6-v2[1, 2, 15]. The *jina model* is a monolingual model that supports 8192 sequence length. It has a vector space of 768, is based on the BERT architecture and has been pretrained on the C4 dataset. *All-mpnet-base-v2* is a sentence-transformer model that uses a 768 dimensional vector space and is provided through the sentence-transformers package. Both *jina-embeddings-v2-base-en* and *all-mpnet-base-v2* were chosen based on their ability to be run as local models, while maintain-

ing statistically good performance. *All-MiniLM-L6-v2* is also a sentence-transformers model, which maps to a 384 dimensional vector space. This model is the default model used by ChromaDB that is automatically applied when storing and querying from ChromaDB, unless otherwise specified.

2.3 Performance Evaluation

Any newly developed system needs to be evaluated through empirical and qualitative studies to ensure that using it is worthwhile. To properly evaluate a system, we need to first define which metrics to look at, why those metrics are important and how they will be measured. In this thesis, the key areas to examine include natural language processing metrics, LLM responses, user evaluations, and cost analysis, which are discussed in the following sections.

2.3.1 Natural Language Processing Metrics

In the field of natural language processing (NLP), tokenization is an essential step to process raw text into smaller units called tokens to train models and evaluate the results [25]. This can be done in a variety of ways such as breaking down text into words, subwords or characters. Tokens is one of the most used metrics in calculating efficiency and cost in NLP applications.

Natural language processing evaluation depends heavily on what type of task the model has been given, including text classification, sequence-to-sequence tasks, language modeling, text generation, and question answering. Text generation and question answering are the two categories that relate to generating coherent and accurate text within a given context, such as asking a chatbot a specific question. Text generation can be evaluated using BLEU and ROUGE scores, which are algorithms that evaluate the quality of the generated text [27, 24]. This, however, is usually difficult to examine even with these algorithms, as it often needs human input to get a reliable evaluation. Question answering is closely related to text generation and is usually evaluated based on correctness and completeness using the F1 metric [29]. These provide a method to empirically measure the performance of natural language processing. However, one of the challenges with these methods is that one needs to have a set of correct answers and texts to compare, which is not always available.

2.3.2 LLM Response Evaluation

The main aspects of LLM responses to be evaluated can be generalized as accuracy and cost. In order to evaluate LLMs, there are various metrics to be observed and analyzed, including natural language processing metrics, LLM-as-a-Judge methods, and human analysis of responses. While it is possible to measure certain metrics with precision, some metrics related to accuracy, such as hallucinations and instruction adherence, are much more difficult to measure with objectivity.

One way to evaluate LLMs accuracy is to use another LLM to analyze responses. The concept of using LLM-as-a-judge comes from the fact that in order to evaluate the response of an LLM, there needs to be some sort of intelligent actor involved, since it can be difficult to implement accurate and reliable algorithms to evaluate responses. This can be done via humans who rate how good a given response is, or can be done by another LLM that generates

a rating of responses. While this raises the question of how well the LLM judge would evaluate the responses, it can also become unfeasible to conduct human analysis on all responses when the dataset grows significantly [8].

Another way to evaluate the accuracy of LLM responses is through human evaluation, where humans create a "gold standard" reference that the LLM responses can be compared to. The term gold standard stems from medical literature and describes a reference test that is without errors to which different diagnostic tests can be compared to [12]. Similarly, this gold standard reference can be used to evaluate LLM responses. This is, however, the most expensive and time consuming method of evaluating LLM responses and is often difficult to do at a large scale. At the core, there is a need for humans to provide correct answers that the LLM response can be compared to in order to assess response accuracy.

2.3.3 User Evaluation

An important part of product development is testing with real-world users to make sure that it functions as intended and efficiently solves the target problem. It is common for developers to have a preconceived idea of how the product is supposed to be used, which can sometimes mismatch with user expectations. To better understand how the product should be designed, developers conduct user tests where potential users are given scenarios in which they use the product to solve a given problem [28]. During these tests, a moderator would guide the tester through the scenario without giving away anything about how to solve the problem. Meanwhile, one or more observers take notes on what the tester is doing, which includes anything from where they are clicking to in which way the participant solves the task. It is also common to apply the *think-aloud method* during these tests. This is a method in which the tester is asked to say out loud everything related to solving the task while they are interacting with the system. By doing this, the observers can gain more insight into why the tester is doing certain things, which further helps in understanding how the tester wants to use the product.

It is also common to conduct a debriefing after the test which usually consists of an interview and a post-test survey to gain insights into the user experience in interacting with the system. A common way to do the post-test survey is using a Likert scale questionnaire. This is a tool which prompts the user to rate how they feel about a set of statements from strongly disagree to strongly agree. By doing this, researchers can turn feelings and attitudes of participants into quantifiable data, which can be analyzed and visualized how users feel about the system.

2.3.4 Cost Analysis

Cost-benefit analysis in the traditional sense looks at how much a given project costs in contrast to how much it can benefit some group or organization. This can be widely applied to a range of different types of projects, such as government undertakings or companies developing a new product. In its simplest form, a cost-benefit analysis calculates how much a project will cost to implement and compares that to the projected revenue or savings from that project. It is worth noting that the benefits do not necessarily have to be monetary, but can also include quality of life and other intangible factors [19].

From a business perspective, there are three main steps are required to analyze a project and determine if it is worthwhile to invest in. First, one is to identify the expected costs and benefits, then to analyze the expected relationship between these costs and benefits, and finally to decide if one should invest in the project or not [21].

A simple way to do a cost-benefit analysis on an internal system is to estimate the time saved per task when using the system compared to the cost of operating that system. These measurements together with personnel costs can be used to calculate a cost-benefit ratio that can be used to justify the use of the system. These estimations are not always easy to produce. Therefore, it is recommended to be conservative in the analysis, using low-end estimates for the benefits and high-end estimates for the costs in order to avoid false positive cost-benefit ratios [19].

2.4 Related Work

Software developer onboarding remains a challenging process, particularly due to difficulties in understanding large codebases, navigating documentation, and receiving timely support. Prior research shows that newcomers often struggle with software comprehension, unclear or outdated documentation, and identifying relevant tasks, which delays their ability to contribute effectively[22]. These challenges motivate the development of automated support systems that can provide contextual and on-demand assistance.

Recent work has explored the use of LLMs to support onboarding. Larsen et al. propose an LLM-based assistant designed to answer project-specific questions and support developers in understanding software systems. Their findings indicate that such tools can assist with summarizing documentation and explaining code, but are limited by their reliance on pre-trained knowledge and lack of project-specific context [22]. This highlights the need for approaches that better integrate contextual information from the target system.

To address these limitations, Ionescu et al. introduce a multi-agent onboarding assistant that combines LLMs with RAG and chain-of-thought reasoning. Their system provides dynamic, context-aware support by retrieving relevant project artifacts and generating tailored explanations, reducing reliance on human mentors. Initial evaluations show promising results in improving onboarding efficiency, although the study is limited in scale and requires further validation in real-world settings [18]. Building on these works, this thesis investigates how similar techniques can be applied and evaluated in an industrial context, with additional focus on system design and efficiency.

Chapter 3
Methodology

To address our research questions, we followed the Design Science Research (DSR) approach. This includes problem conceptualization, solution design and evaluation [13]. To begin with, we set out to explore problems with the technical integration part of the onboarding process in the industry and at Bosch. A prototype solution was designed using modern LLM techniques and designs. Finally, we evaluated the solution by conducting user evaluation tests and gathering empirical data to estimate the cost of using the system. Figure 3.1 shows an overview of our thesis process, inspired by Noah Mayerhof and Sandra Nyström [26].

3.1 Problem Conceptualization

We initialized the project by identifying the current challenges in the technical integration process through literature reviews and a case study. We analyzed the current practices both in the industry through research, and at the case company by conducting semi-structured interviews to find out the problem areas. We also studied current LLM techniques to familiarize ourselves with their capabilities and evaluation methods to find out what approaches would be best suited to implementing a solution for our identified problems.

3.1.1 Literature Review

We conducted a literature review to investigate the technical integration part of the onboarding process as well as modern LLM techniques. For gathering relevant literature, we used *Finn (Lund University)* and *Google Scholar*, which both contain a large index of databases. *Finn* has a more narrow reach, but uses generally more reputable sources, while *Google Scholar* has a larger index database, but also a varying mix of reputability in their sources. When conducting our searches regarding methods and techniques we prioritized peer-reviewed papers, but also included selected technical documentation (e.g., frameworks and tools) where relevant to current industry practice.

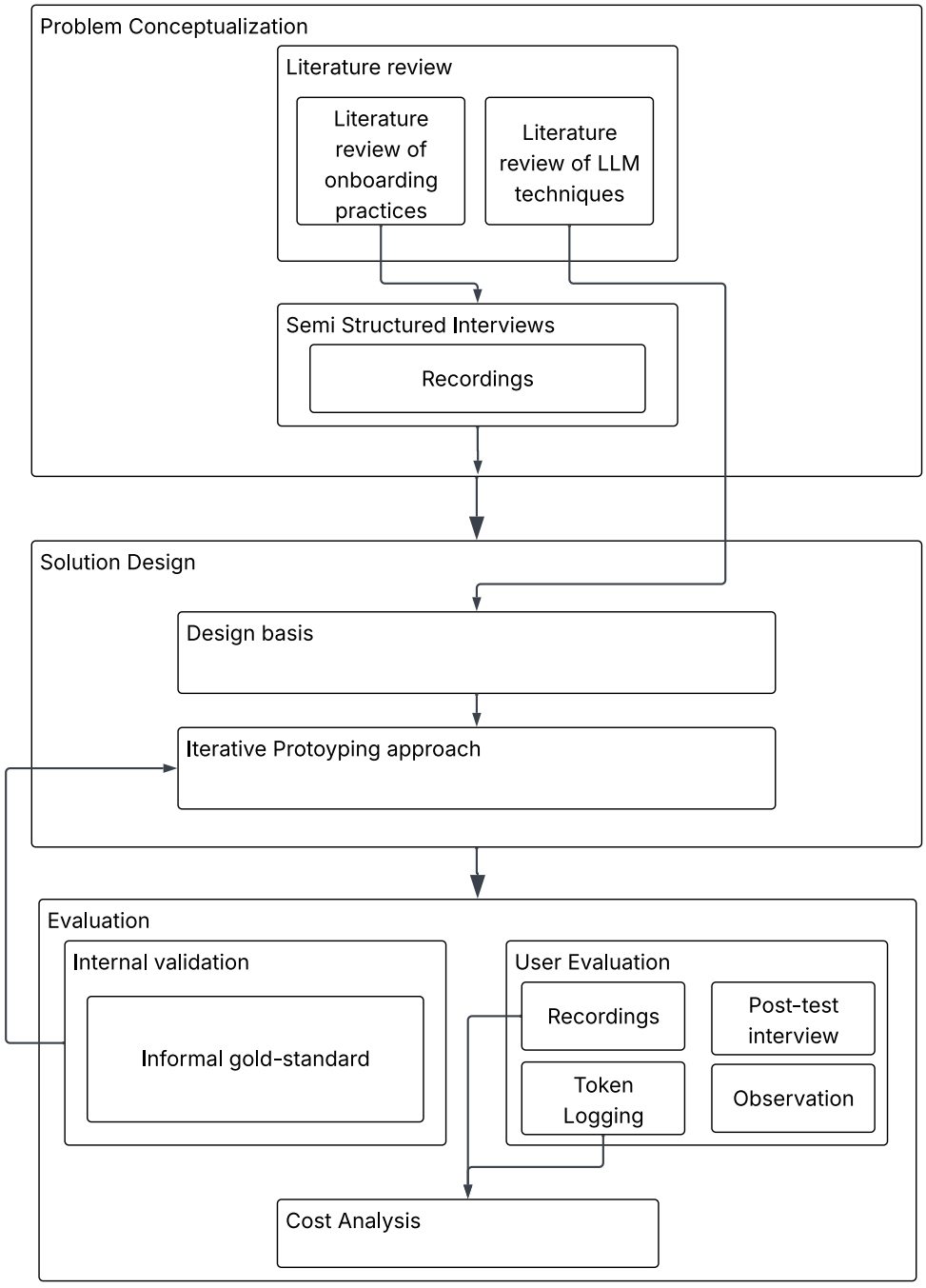


Figure 3.1: Overview of thesis process

For the literature review investigating the technical integration part of the onboarding process, the major keywords we searched for were *software onboarding*, *AI onboarding assistant*, *onboarding strategies*. Articles were included if they addressed software onboarding processes or AI-assisted onboarding in organizational contexts. Articles were excluded if they focused on non-software parts of the onboarding process. The most relevant studies we found include Buchan et al. [11], Ju et al. [20] and Bauer, Erdogan [9] this is covered in Section 2.1.

For the literature review investigating modern LLM techniques, the major keywords we searched for were *RAG*, *Agentic systems*, *LLM evaluation*. Articles were included if they addressed AI-assisted onboarding, presented methods, architectures, or evaluation approaches related to large language models, such as RAG or agent-based systems. Articles were excluded if they were not directly related to LLM-based systems, lacked methodological detail, or originated from non-reputable or non-verifiable sources. The most relevant studies based on these criteria we found include Alabdulwahab et al. [8], Lewis et al. [23], Huang et al. [16]. Here we also studied online resources such as ChromaDB [3], LangChain [4] and PocketFlow [5]. This is covered in Section 2.2 and Section 2.3.

The selection process was conducted collaboratively by the authors. Initial screening based on titles and abstracts was performed independently, followed by joint discussions to resolve disagreements and reach consensus on inclusion. This iterative process ensured both consistency and relevance in the final set of literature.

3.1.2 Semi-structured Interviews

The current technical integration process at Bosch was examined by conducting semi-structured interviews with recently onboarded developers. This was done both to investigate the technical integration at Bosch as well as relating Bosch's practice with well-researched methods.

Semi-structured interviews were performed to gain insights into how Bosch currently handles its technical integration. We explored what is currently working well, what is not and what can be improved. An interview guide was drafted based upon the previous literature review and what we need to know for our solution to be useful for developers. After a draft was created, a pilot interview was held to evaluate if the guide was on the right track and also to give us a time estimate for the interviews. The final interview questions are listed in Appendix B

Participants were then identified through previous contacts, office socialization and interacting with developers new to the team. This gave us a group of participants in various teams working on different projects, as shown in Table 3.1.

The interviews were held with both of us present, with one person guiding the interview while the other observed and asked potential follow up questions to further clarify answers. During the interview a list of task-assignment strategies and onboarding strategies was brought for the participant to look at, so that they could better describe their technical integration process. This list was based on the information presented in Section 2.1 and can be seen in Chapter A. The interviews took about 20 minutes to conduct but there was no time limit set and participants were allowed to elaborate as much as they wanted. These interviews were recorded to allow for analysis after the interviews were concluded.

In order to analyze the interviews, we focused on four main aspects from which we could draw general conclusions about the technical integration part of the onboarding process at Bosch, based on the interviews. These aspects were, an *overview of the technical integration*

process at Bosch, which *task assignment* strategies were used, what *onboarding strategies* were used for the technical integration, and what the main *challenges* were for new developers during their technical integration.

The interviews were analyzed using a thematic analysis approach. First, all interviews were transcribed and reviewed to achieve familiarity with the data. We identified sections from the interviews corresponding to the different aspects. These sections were then analyzed further together to form general conclusions for each of the four aspects. The analysis was conducted independently by the authors, after which the results were compared and discussed to resolve discrepancies and reach consensus.

Role when hired	Previous experience (Years)	Identifier
Embedded developer	2	I1
Software engineer	3	I2
Algorithm developer	Fresh graduate	I3
Junior software developer	Fresh graduate	I4

Table 3.1: Role and experience level of interviewees at time of hiring

3.2 Solution Design

The design of the proposed system, Bob, was carried out using an iterative and research-informed approach, using a literature review and interviews as input. The process combined insights from existing literature, identified challenges from industry practice, and continuous prototyping and evaluation. Rather than defining a fixed architecture from the outset, an initial baseline solution was developed and incrementally refined through multiple design iterations.

The overall design process was guided by two primary inputs. First, relevant research on large language models and retrieval-augmented generation informed the selection of suitable architectural patterns and techniques. Second, challenges identified during interviews with company representatives, particularly difficulties in locating and utilizing internal documentation, providing practical requirements that the system needed to address.

Based on these inputs, a series of design decisions were explored through prototyping, focusing on aspects such as data retrieval methods, system architecture, and model selection. Each iteration was evaluated internally, and the results were used to guide subsequent improvements. This approach enabled the gradual development of a solution tailored to both theoretical best practices and real-world constraints. The following sections describe the basis for the design decisions and the iterative process used to develop the final system.

3.2.1 Design Basis

The design of the system was grounded in a combination of theoretical insights and practical constraints. A key architectural decision was the adoption of a retrieval-augmented generation (RAG) approach. This decision was motivated by existing literature indicating that RAG systems can improve the performance of large language models by incorporating external knowledge sources into the generation process. Additionally, this approach directly

addressed a challenge identified during interviews, where company documentation was described as difficult to locate and utilize effectively. By integrating a retrieval mechanism, the system could dynamically access relevant internal information and provide more contextually accurate responses.

Several practical constraints also influenced the design. Due to hardware and security limitations, the system was required to run locally on company machines, with the exception of the language model itself. This constraint guided the selection of tools and technologies, favoring solutions that could be deployed and executed in a local environment. For example, a vector database with minimal setup requirements and local hosting capabilities was prioritized to support efficient document retrieval.

The system architecture was shaped by the decision to adopt an agent-based design, allowing for modularity and flexibility. This modularity helped design components that could independently interact with our agents. An additional motivation this decision was the use of smaller and more cost-efficient language models, which benefit from operating on limited context windows. By structuring the system as a set of specialized agents, each component could operate on a reduced context, improving efficiency while maintaining overall system performance

During the design process, multiple frameworks and tools were considered to support the implementation of the system architecture. Pocketflow, a graph-based framework was selected due to its ability to clearly represent and manage complex agent flows, while remaining relatively simple to use and extend. Conversely, alternative approaches based on MCP architecture were evaluated but ultimately not adopted, as they lacked suitable support for the specific use case and available infrastructure.

Together, these considerations combines insights from literature, empirical findings from interviews, and practical system constrains to form the foundation for the subsequent iterative design and development process.

3.2.2 Iterative Prototyping Approach

The system was developed through an iterative prototyping approach, in which an initial baseline solution was incrementally improved. Rather than attempting to design a complete system upfront, a minimal viable prototype was first constructed to establish core functionality, including basic data retrieval, question processing, and response generation.

Following the establishment of this baseline, the system was refined through a series of iterative design cycles. Each iteration focused on improving specific components of the system, such as the data retrieval process, the structure of the agent-based architecture, and the selection of models used for embedding and text generation. This allowed individual aspects of the system to be explored and optimized independently while maintaining an overall functional prototype.

3.2.3 Evaluation-Driven Iteration

The iterative design process was guided by continuous internal validation, where intermediate prototypes were systematically assessed to inform subsequent design decisions. Validation was primarily conducted by the authors through structured testing of individual system components, focusing on their ability to perform intended tasks effectively.

Different components of the system were validated using task-specific criteria. For the data retrieval component, embedding models were assessed by comparing the relevance of retrieved documents for identical queries, with particular attention to whether key documents were successfully identified. This enabled a comparative assessment of how well different models supported accurate information retrieval.

Language models were evaluated based on the quality and consistency of generated responses. In cases where multiple models produced comparable outputs, selection was guided by practical considerations such as computational efficiency and cost. This ensured that the chosen model met performance requirements while remaining feasible within the project constraints.

The agent-based architecture was refined through empirical testing of system behavior. In particular, scenarios where the system failed to retrieve or utilize relevant information were used to identify limitations in the existing design. These observations motivated modifications to the agent flow, enabling improved handling of such cases in subsequent iterations.

The validation process was primarily internally driven, with limited external feedback during the development phase. However, the continuous assessment of intermediate prototypes ensured that design decisions were grounded in observed system performance rather than assumptions. A more detailed description of the evaluation methods and results is presented in Section 3.3.

3.3 Evaluation

An important part of our thesis was to continuously evaluate Bob from development through the final stage. This was done both through performance observations during development and user evaluations of the final system. In order to ensure that the user evaluations were conducted on the best possible version of Bob, internal validations were made continuously during implementation. This approach allowed less optimal solutions to be discarded without having to spend large swaths of time on formal evaluations of multiple implementation variations.

3.3.1 Internal Validation

Our performance evaluation for Bob was done mostly through internal validations. This was done due to time constraints since we could be more agile in our work and change things as we went along, which would not have been possible if formal tests had to be set up for each iteration. We also concluded that advanced metrics related to hallucinations and instruction adherence were not necessarily the most important to evaluating our proof-of-concept system. These internal evaluations were conducted by ourselves by continuously testing the system with different prompts as we implemented it. We were able to directly evaluate the answers from Bob, as we had full access to the repository and could verify if those answers were truthful and adhered to our instructions. In this way, we created an informal gold-standard evaluation by establishing what a correct answer should look like and making sure that the answers from Bob were correct. For example, if asked to summarize a given project, Bob should pick out the main aspects of that project without hallucinating any parts that are not present in the project. We will then ourselves check to make sure that not part of

the answer is hallucinated or focuses on the wrong things. We also roughly take note of how long it takes to generate an answer to ensure that our solution is still able to run in a timely manner. Our aim for this was set at no more than 15 seconds.

3.3.2 User Evaluation

We have conducted user tests to evaluate how well Bob works to assist developers with their learning. In accordance with the methodology described in Section 2.3.3, we set up test environments where developers at the case company were given tasks to solve with the help of Bob. The developers were first introduced to our project and the purpose of Bob. They were then presented with a project that was unfamiliar to them, with no further explanation as to that projects function, and were assigned some simple tasks to evaluate the performance of Bob for different use cases. Based on our semi-formal interviews we defined a set of areas for a thematic analysis that would guide the design of our tasks for the user tests. These areas were *Information search (documentation database)*, *Information search (repository)*, *Solving code specific tasks*, *Increased task efficiency* ,*Ability to act as colleague/mentor*, *Overall experience*. The tasks are presented in Table 3.2. Similarly to our semi-formal interviews, participants were sampled by finding relatively new developers in the Bosch office, some of which were also participants in the semi-formal interviews.

Task no.	Task description
Task 1	Give an overview of what this program does
Task 2	Explain what you need to do in order to set up the program and start streaming data
Task 3	Find the release page for this team in the documentation database
Task 4	Find out what the test scope and test status is for this project
Task 5	Find the standard guide for setting up a python project for this team
Task 6	Find out how many answer types there are defined in this project
Task 7	Suggest a fix to a bug that we have introduced to the code of this program

Table 3.2: Tasks that were assigned during user tests

During the test, one person acted as a moderator, who guided the user through different tasks. The test was also set up on this persons personal computer which was recording the whole test. The other person acted as an observer who took notes of everything that the tester was saying and doing during the test. This included things such as which information from Bob they focused on, which files and pages they looked at based on Bob's references, if they expressed trust in the information they received and how they used that information to come up with an answer. This process was further amplified by the think-aloud method which the participant was instructed to follow during the test. In total the test took around 45 minutes to complete. The moderator would interrupt the tester and ask them to move on to the next task in case they were unable to finish a given task to ensure this time window was kept consistent throughout all tests. This was done at the moderators discretion when the user was unable to make meaningful progress, and it was reasonably clear that they would not be able to finish the task.

Once the test was finished, a post-test interview was conducted where we asked questions about the users' experience in using Bob (see Table 3.3). The questions were designed to let

the tester speak freely about their overall experience and also connect their experience to certain technical integration aspects.

Question no.	Question
Q1	Did you find Bob helpful when completing these tasks?
Q2	Do you think you have completed the tasks faster without Bob?
Q3	Would you have preferred asking a colleague fo help instead of asking Bob?

Table 3.3: Post-test interview questions

Finally, the testers were given a short survey in which they got to rate how much they agreed with different statements, from strongly disagree to strongly agree in Likert scale. These questions can be seen in Table 3.4. In order to compile the qualitative data that was gathered from the user tests and the post-test interview, we performed simple thematic analysis was performed that summarized the user tests based on common themes that were observed during the tests. The main areas that were observed were the following: if information was properly fetched and summarized, if specific task solutions were proposed, if tasks were solved faster with Bob, if it would have been preferred to ask a colleague for help, and if the overall experience was that Bob was helpful. These areas were evaluated based on what the user said during the test and subsequent interviews, measured as positive, neutral or negative. This was then used in conjunction with the quantitative data from the surveys to analyze the results.

Question no.	Question
Q1	Bob helped me solve most of the tasks.
Q2	Bob was easy to use.
Q3	Bob helped me find relevant Docupedia articles.
Q4	Bob provided seemingly correct information.
Q5	I think I could have solved the tasks faster without Bob.
Q6	Bob found the code snippets that I was looking for.
Q7	I think Bob could guide me similarly to how a senior developer would.
Q8	I would recommend Bob to new developers.

Table 3.4: Post-test survey questions

Test Environment

The tests were carried out on the same machine with a screen recorder running to capture the tests. The repository on which the tasks were given was presented in VS code with basic extensions enabled. Copilot was disabled for the tests, since its use could interfere with the evaluation of Bob to a great extent. Bob itself was brought up in a Firefox browser window on a second screen, since this is the most practical way of using Bob while coding simultaneously. An office room was booked for all tests to limit disturbances.

3.3.3 Cost Analysis

We conducted a cost analysis on the utilization of Bob to estimate how much it would cost to use it in everyday operations. From our user evaluations, we logged data on how many tokens were used when doing simple tasks that are likely to be similar to what a new developer would be doing, as well as how much time was spent doing the tasks by timing when the test was started and finished. Here we also split up token usage percentage wise depending on if the tokens are input tokens or output tokens, since the cost of these are slightly different. This was then extrapolated into the token usage over one hour, where we calculate the cost per hour for a developer to use Bob, as given in Equation (3.1). The cost of input and output tokens are taken from the internal documentation at Bosch which lists token cost depending on the chosen LLM model.

$$cost_{tot}/hour = tokens/h * percentage_{input} * cost_{input} + tokens/h * percentage_{output} * cost_{output} \tag{3.1}$$

This calculation is taken into account when discussing the value of using Bob (see Section 5.3) when comparing to the perceived usefulness of Bob, which can be seen in our qualitative data from the user evaluations Section 4.3.

Chapter 4
Results

In this chapter, we present the results obtained from investigating each research question posed at the beginning the thesis. We present common strategies for the technical integration part of the onboarding process and challenges with that process at Bosch (RQ1), propose an LLM based solution design that addresses these challenges (RQ2), and show our evaluation of this solution based on user evaluations and a cost analysis.

4.1 Strategies and Challenges in the Current Process (RQ1)

The current technical integration process was investigated in two steps described in Section 3.1, containing a literature review and developer interviews. The literature review yielded insight into developer onboarding, general onboarding methods, common challenges, and task assignment strategies. This provided a foundation for formulating our interview questions, which covered topics such as the most challenging aspects of onboarding, the participants' first task and how well the technical integration prepared them, as well as background information about the participant and their introduction time.

Our findings from the literature review are presented in Section 2.1. We identified a number of technical integration strategies as well as three key task assignment strategies, which guided the question formulations for our semi-structured interviews. The strategies are listed in Appendix A.

The findings from the semi-structured interviews are presented in Table 4.1, where we have highlighted the main points from each of the four aspects that we focused on analyzing from the interview. All participants said that there was no formal technical integration process for them, and while they did recognize some of the strategies that we presented them with those were usually not applied intentionally.

The most common task assignment strategy was simple-complex. All participants re-

called that they were given some easier task to start off with and then slowly moved on to more complex tasks as they gained confidence in their abilities. One participant mentioned that he had been put onto a new project in which a priority-first strategy had been used but at that point he was also a bit more experienced with the company.

> "My first tasks were simple bugfixes" -I4

Technical integration overview
• No formal technical integration • Variety of strategies used, both intentional and unintentional
Task assignment
• Simple-complex most common • Priority first when needed • Exploratory rare
Strategies
• Used informally • Mentoring / peer support • Self learning • Knowledge database
Challenges
• Finding documentation • Solving niche technical problems • Setting up workspace, installing dependencies etc.

Table 4.1: Summarized results from semi-formal interviews

Two participants (I1, I2) expressed that they were onboarded using the mentoring strategy, where a mentor helped the new developer with getting to know the project. These two cases differed slightly as one participant had a mentor formally assigned that focused on teaching specifics of the project while the other participant had a semi-official mentor that assisted with more general tasks.

> "I did not have a specific person as my mentor but I usually asked the same person in the team for help" -I1

Closely related to this, all participants mentioned that peer support was a significant part of their onboarding, and they often found themselves asking other team members for help. This was sometimes close to becoming instances of informal mentoring, as participants often went to the same team member for help. However, most participants said that they asked whoever had the most expertise in a given area for help with more niche problems. All participants also said that they practiced self learning and use of the Bosch knowledge database Docupedia.

During all interviews, a common pain point was either the lack of documentation or that the correct documentation was hidden away in the massive documentation database

Bosch uses. Setting up the correct development environment, getting access to essential development infrastructure and finding documentation were things that all participants found frustrating. These activities fall under the knowledge database onboarding strategy at Bosch. However, it was clearly indicated that there were several major flaws with this system.

> "When i started here I always had to ask someone else to help me find the docupedia pages since they were all so nestled into each other" -I2

Other onboarding strategies were used by the participants to varying degrees. A product overview was given to some at the start of their employment, however, one of the participants who received this noted that it was nice but not that effective at helping them gain a deeper understanding of the product. Team meetings were a common occurrence for all participants; however, multiple participants mentioned that this was more of a run-of-the-mill activity and not intended to specifically help new developers onboard to the team. Online search and self-learning were mentioned as supplementary activities to reading and learning from the documentation database. The AI chat tool at Bosch was sparsely used or not available at the time when the participants were doing their onboarding, and no participant mentioned pair programming as an onboarding strategy.

In general, the technical integration process at Bosch is quite informal, teams decide how to conduct the technical integration process themselves. In common, all participants used some form of self-learning, including reading code, documentation reading, and online search depending on the work assigned. It took between 3 to 6 months for developers to become comfortable with development. The most common problem was a lack of easily accessible documentation or missing documentation. The most common forms of external help were either mentoring or peer-support, depending on whether a mentor had been assigned.

4.1.1 LLM Practices

While not pertaining specifically to onboarding strategies and thus our first research question, we also conducted parts of the literature review on current LLM techniques and strategies that we could apply to create Bob. The findings from this part of the literature review are presented in Section 2.2 and we have based our tech stack on our findings.

4.2 Bob Implementation (RQ2)

This section presents the final design and implementation of Bob, the proposed system for supporting developers in navigating project-specific knowledge. The section provides an overview of the system architecture and describes its main components, including the data pipeline, retrieval mechanism, and agent-based processing flow. The aim is to clearly outline how the system operates and how its components interact to produce responses to user queries.

4.2.1 System Overview and Usage

Bob is designed as a RAG, agent-based system that assists developers in answering questions related to specific software projects. The system integrates multiple components, including a

data pipeline(Data Loader), a vector database (ChromaDB), and an agent flow (Pocketflow), which together enable the retrieval and generation of contextually relevant responses.

The system operates in two main phases: an initialization phase and a query phase. During the initialization phase, a selected project repository and its associated internal documentation are collected, processed, and stored in a vector database. This process includes extracting relevant data, segmenting documents into smaller units, and generating vector embeddings. As this step involves processing large amounts of data, it is performed infrequently, typically when a new project is loaded or when significant updates occur.

In the query phase, which is executed for each user interaction, the system processes a user question through an agent flow. The baseline flow can be seen in Figure 4.1 which processes a user question(GetQuestion) into a RAG query(Search-Optimization, Query) and produces a final formatted output(Answering, Formatting). The agent flow is what changes output based on the graph structure and input of the user. The agent flow will be more thoroughly covered in Section 4.2.4. Lastly an overview of the system architecture and component interactions is illustrated in Figure 4.2

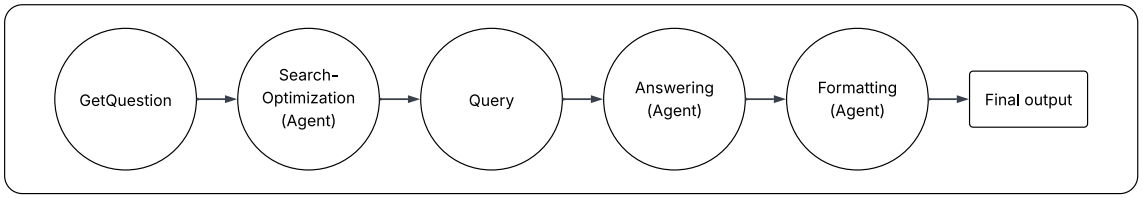


Figure 4.1: Baseline Agent Flow

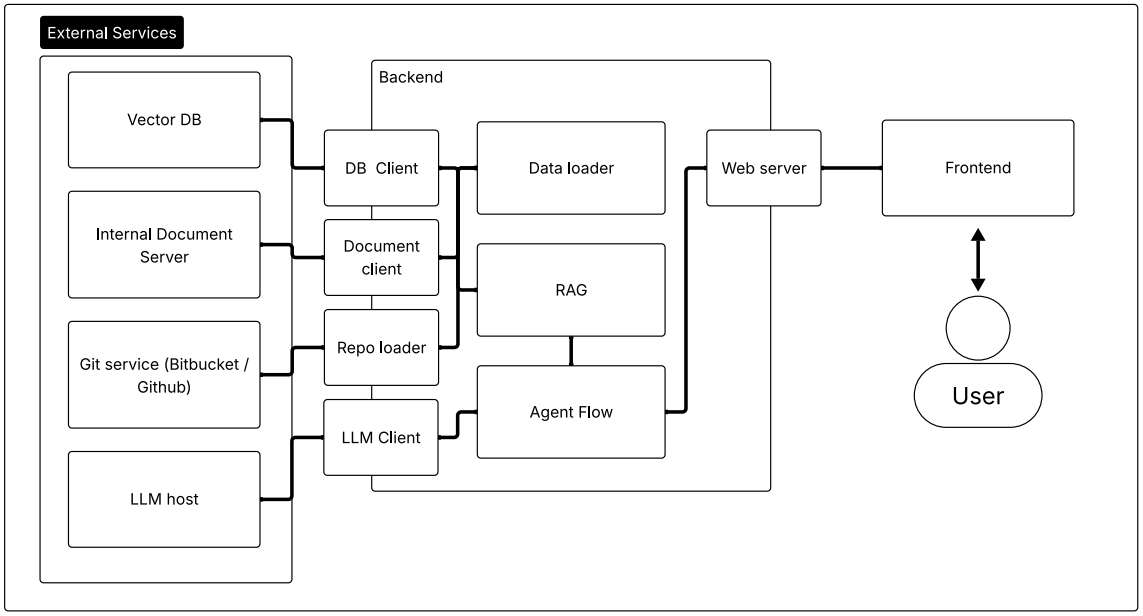


Figure 4.2: Baseline Architecture

4.2.2 Data Pipeline

The data pipeline is responsible for collecting, processing, and storing information used by the system for retrieval. The system integrates data from two primary sources: project repositories and internal documentation.

Data is collected by extracting files from the selected repository and retrieving related documentation through an internal documentation API. To improve relevance, repository metadata is used to identify associated documentation, enabling targeted data collection.

Once collected, documents are processed by dividing them into smaller segments to facilitate efficient retrieval. These segments are then transformed into vector representations using an embedding model and stored in a vector database. The database is structured into separate collections for different data types, such as source code and documentation, allowing more precise querying.

The decision for designing it this way was made based upon the data size which is not exactly known, approximately 3000 to 7000 documents are added, and 40000 pages are updated per day. We, therefore, found that the most valid way to gather data was to fetch and process in real-time. To do this, we implemented a vector database using ChromaDB that could be queried using semantic search queries.

4.2.3 Retrieval and Context Integration

To generate relevant responses, the system retrieves context from the vector database based on the user prompt. Rather than directly using the user input, the system first generates a refined query representation using agents(SearchOptimization), which improves the effectiveness of the retrieval process.

This query is used to perform a semantic search in the vector database, returning a set of documents ranked by similarity. To ensure relevance, retrieved documents are filtered based on their distance in the embedding space, limiting the context to the most relevant results.

To manage computational cost and ensure efficient use of resources, the system restricts the amount of context provided to the language model. A predefined token limit is applied to each request, ensuring that only the most relevant information is included.

In addition to repository and documentation data, the system also incorporates prior interactions by storing previous question–answer pairs in the vector database. These are retrieved in the same manner as other documents, allowing relevant conversation history to be included when applicable, without requiring the full history to be processed for each query.

4.2.4 Agent System and Prompting Strategy

The system employs an agent-based architecture to manage the processing of user queries. The workflow is structured as a sequence of interconnected nodes, where each node performs a specific task within the overall processing pipeline.

The agent flow begins by receiving a user query, which is then processed and transformed into a form suitable for retrieval. Based on the query, relevant documents are fetched and passed to an answering agent, which generates a response using the provided context. The response is subsequently formatted before being returned to the user.

The architecture was extended with additional agents to improve system performance and flexibility. For example, specialized agents were introduced to determine whether retrieval is necessary for a given query(Breakdown) and to refine how information is gathered(Local File Search). In some cases, additional steps are conditionally executed to balance performance and computational cost.

Prompting plays a central role in guiding agent behavior. Each agent is provided with structured instructions that define its task and constraints. These prompts are designed to ensure that agents perform focused operations, such as retrieval, reasoning, or formatting, contributing to a modular and controllable system design.

An overview of the final agent flow is shown in Figure 4.3.

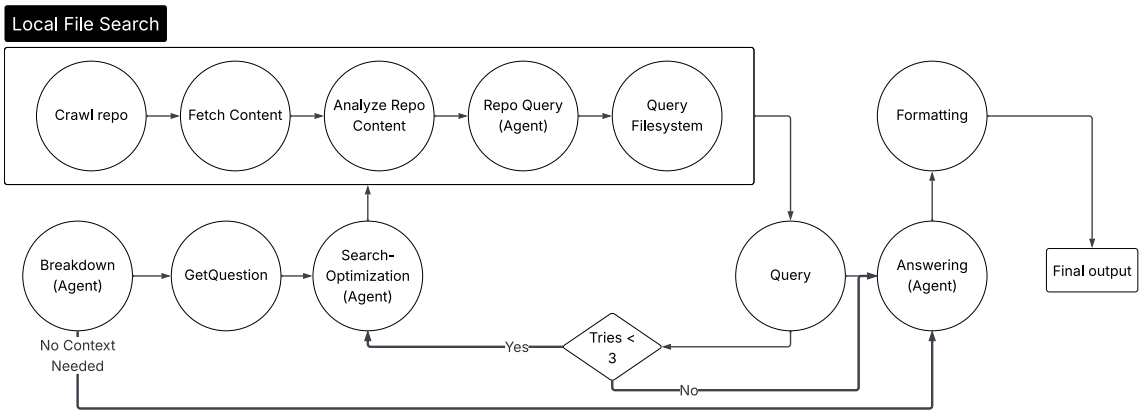


Figure 4.3: Improved node-agent flow

4.2.5 Model and Design Choices

The final system design reflects a set of model and architectural choices made to balance performance, efficiency, and practical constraints. For document representation, an embedding model was selected that provided effective semantic retrieval while being suitable for local execution. This enabled efficient querying of the vector database without requiring external infrastructure.

For response generation, a lightweight language model was chosen(Gemini 2.5 flash lite), as it provided comparable output quality to larger alternatives while offering improved response times and lower computational cost. This was particularly important given the goal of developing a system that could scale in practical use.

From an architectural perspective, an agent-based design was adopted to enable modular processing of user queries. This approach allows individual components to operate on smaller, task-specific contexts, improving both efficiency and controllability when working with cost-constrained language models.

Additionally, all components of the system, except for the language model, were designed to run locally. This decision was driven by practical constraints related to available hardware and data access, ensuring that the system could be deployed within the target environment without reliance on external services.

Overall, these design choices contribute to a system that balances retrieval accuracy, response quality, and computational efficiency within the constraints of the application con-

text.

4.3 Bob Evaluation (RQ3)

The final step of our project was to evaluate the final version of Bob. In this section, we present our experimental result.

4.3.1 User Evaluation

The testers were all developers from Bosch, four of which had roughly two years experience working at the company and one who was a student worker without much experience (see Table 4.2). Three of the testers were also part of our semi-formal interviews that we conducted at the beginning of our project. None of the testers had any experience with the repository that we conducted our user evaluations.

Role when hired	Previous experience (Years)	Test Identifier
Software Engineer	3	T1
Algorithm Developer	Fresh graduate	T2
Junior Software Developer	Fresh graduate	T3
Software Engineer	1	T4
Student Worker	Student	T5

Table 4.2: Participant characteristics

We conducted five tests in total with all users performing 6 tasks Table 3.2. In our evaluations, we focused on the main areas presented in Table 4.3. The observations were recorded and written down. Once all tests were completed, we summarized the results to analyze each theme individually. Figure 4.4 shows the overall user sentiments for each analyzed areas. We have also included a few quotes that represent the sentiment for each area.

Numbering	Area
Area 1	Information search (documentation database)
Area 2	Information search (repository)
Area 3	Solving code specific tasks
Area 4	Increased task efficiency
Area 5	Ability to act as mentor/colleague
Area 6	Overall experience

Table 4.3: Areas analyzed during user evaluations

4.3.2 Thematic Analysis of the User Evaluation

Data gathered from the observations, recordings and post-test interviews were congregated to be the basis for the thematic analysis, which was done to obtain concrete data points for

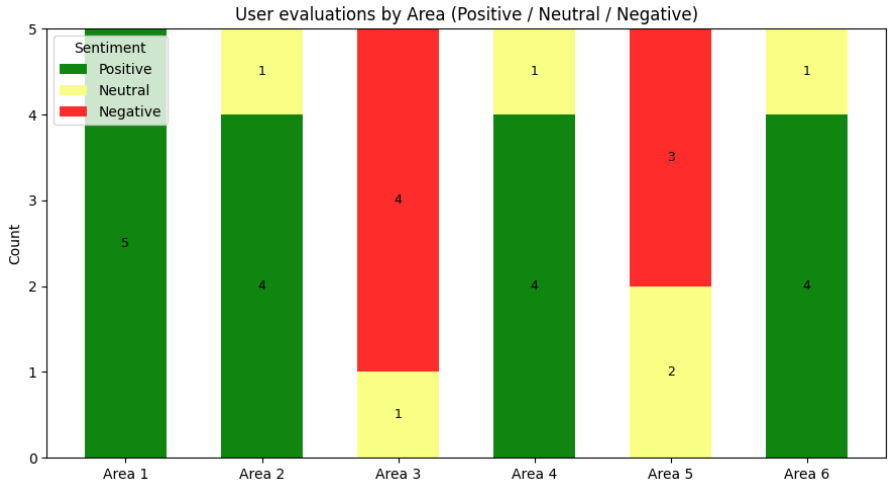


Figure 4.4: User sentiment for analyzed areas

the user evaluation based on qualitative data measurements. The success rate for each task is presented in Figure 4.5. The figure shows that all users were able to complete tasks 1, 3, 4 and 5. One tester did not manage to solve task 2, two testers failed to complete task 6, and all testers failed to solve task 7. In the case of task 7, it is worth noting that Bob successfully guided all testers to the part of the code where a bug had been introduced; however, once there, no tester was able to successfully identify what was actually wrong with the code.

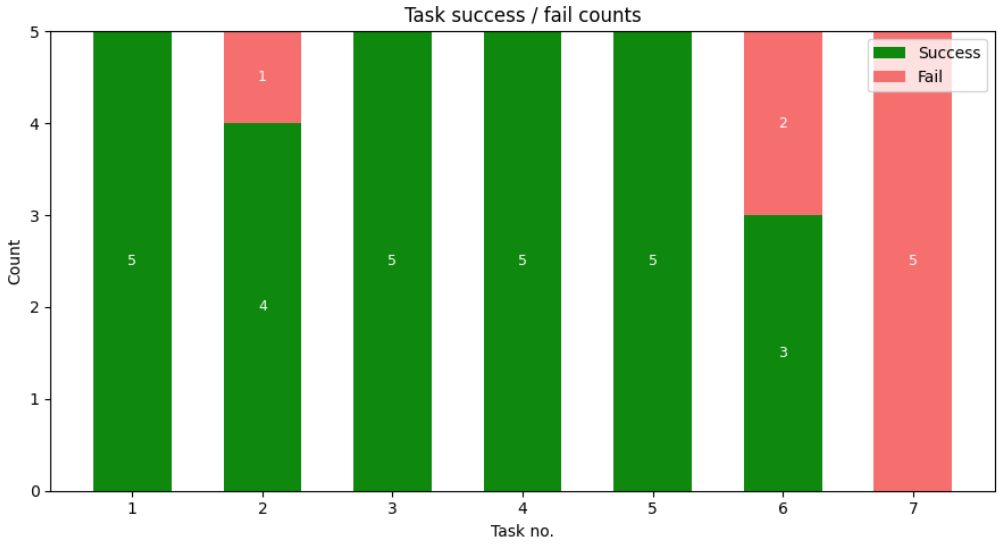


Figure 4.5: Success rate per task conducted in user tests

Information Search

All users expressed a positive experience with the information searching capabilities of Bob. Both the ability to summarize information fetched through RAG and the ability to link to relevant pages and files was viewed positively by all testers. Many expressed that this was

a common pain point in their daily work. Multiple users mentioned that this is something that is especially difficult when you are a newcomer to the company, since the documentation structure is not very intuitive. Task 1 was useful in highlighting the testers experience in this area, which received the highest number of positive responses.

> "Bob was helpful with finding information from Docupedia (Confluence), it is usually difficult to search and find information there" -T3

Task 7 was also an indicator of Bob's ability to find most relevant information, since it was able to pinpoint the function that contained the issue in every test.

Solving Code Specific Tasks

This area was something that all testers expressed trouble with. While Bob was in some cases able to point in the right direction to help with solving tasks, it was rarely able to actually pinpoint the specific error and code that was relevant to the task. This meant that the users had to figure out the solutions themselves or simply fail to complete the task.

> "Hard to tell if any of these are right" -T2

Bob was responded with useful suggestions in some cases; however, the overall experience with the tests was negative when it came to solving a task. This area was mainly analyzed through the results of tasks 6 and 7, where it could be seen that Bob struggled with making accurate suggestions. In task 6, Bob was often not able to find the specific Python class that was required. In task 7, Bob was very successful in finding the correct function that contained a bug; however, it was not able to assess what that bug was.

Increased Task Efficiency

Increased task efficiency is analyzed through question 2 in the post-test interview. Users expressed that Bob was able to help them solve the tasks faster than they would have otherwise. All users expressed that they usually struggle with finding information quickly, and Bob was able to help them in that regard.

> "In this scenario it was faster with Bob. It can be helpful if you are new to a project environment." -T3

Ability to Act as Mentor

The ability to act as a mentor is analyzed through question 3 in the post-test interview. Most users expressed some concerns in relying on Bob instead of asking a colleague or mentor.

> "You learn more by asking someone else and reading the code on your own" -T5

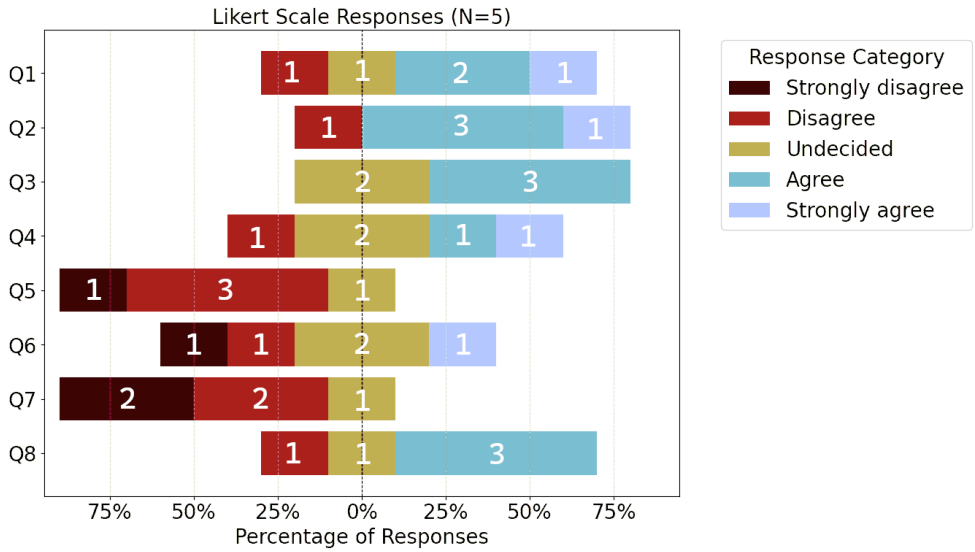
Some users expressed that they are hesitant to ask for help from other developers, since they do not want to waste their time. In such a cases, they would ask Bob first, and if that does not work, then they would ask for help from peers. Still, users agreed that Bob cannot serve as a replacement to asking a colleague or mentor.

Overall Experience

Overall experience is analyzed as a summary of the user test together with the post-test interview, evaluating the user's experience using Bob. Figure 4.6 shows the results of the post-test questionnaire. Users' overall experience was positive, and Bob was particularly helpful when solving certain tasks, such as information search and summarization.

> "It works well in this scenario" -T4 in relation to the test scenario as a whole

Some common observations during the tests were that users tried to solve tasks not simply by relying on Bob, but also searching for information on their own simultaneously. This meant that Bob often served as a helpful tool and not the only solution to the tasks. Bob was deemed helpful as long as it did not hinder the tasks.



Q1 - Bob helped me solve most of the tasks.
Q2 - Bob was easy to use.
Q3 - Bob helped me find relevant Docupedia articles.
Q4 - Bob provided seemingly correct information.
Q5 - I think I could have solved the tasks faster without Bob.
Q6 - Bob found the code snippets that I was looking for.
Q7 - I think Bob could guide me similarly to how a senior developer would.
Q8 - I would recommend Bob to new developers.

Figure 4.6: Post-test survey responses

4.3.3 Cost Analysis Evaluation

Initially, we carried out simple tests that measured time taken and tokens used when running the baseline flow of the system. Based on the Equation (3.1), the cost is estimated to be **€0,017** per hour and **€ 29,83** per year per developer, if a developer used it every working day for 8 hours a day in the same capacity, making the cost very small for the company.

The token usage per test and the cost calculated for that are given in Figure 4.7 and Table 4.4. Where Figure 4.7 depicts the total amount of tokens used in each test, split between input/output. Notable here is that the tester in test 1 used a significantly more tokens. From the recordings of the tests it is clear to see that this person was a bit less patient when getting responses from Bob and would often immediately ask follow up questions instead of fully reading through the response which meant a higher prompt frequency. This was also in combination with a slightly longer runtime than other tests due to an early start of the test. With that said, there were no other outliers in the user evaluation from this test and this token usage simply reflects a certain use case that does not impact the percieved usefulness of Bob. Table 4.4 contains cumulative testing metrics, such as hours for all tests and total tokens used in all tests.

Metric	Value
Hours (test)	2.33h
Total Tokens Used (test)	302323
Tokens / hour (test)	129752
Percentage Input (test)	90.21%
Percentage Output (test)	9.79%
Working Days / year	260
Cost per 1M tokens input	€0.084*
Cost per 1M tokens output	€0.34*

*Conversion USD to EUR fetched 2026-01-26

Table 4.4: Token usage and cost statistics of the user test

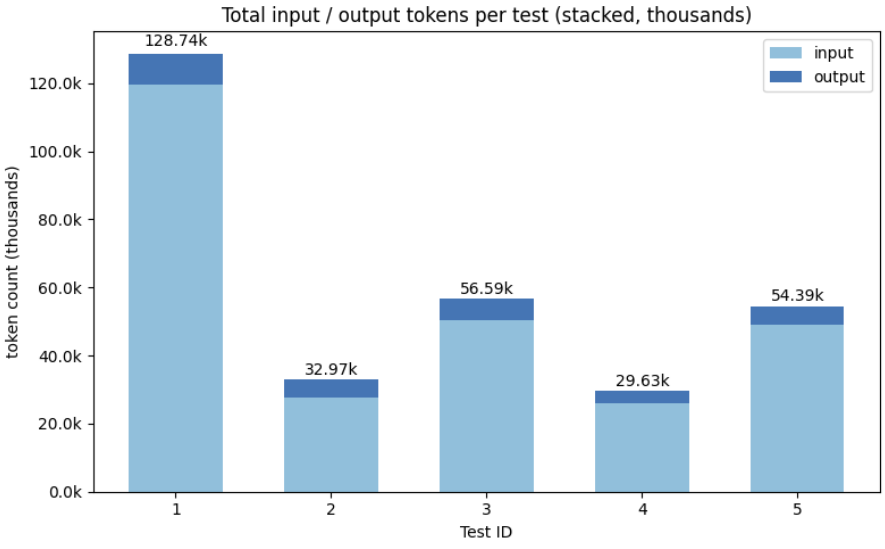


Figure 4.7: Token usage per test

Chapter 5
Discussion

In this chapter, we discuss our findings in relation to each research question. We look at the implementation choices and the development process to evaluate what challenges were successfully tackled, what issues still exist, as well as other problems that could be addressed in future work. We also discuss threats to validity to ensure that our findings are credible and to encourage further research where these threats can be eliminated.

5.1 Challenges in the Current Process (RQ1)

Our case study through a literature review and interviews has provided us with a lot of insight into how the technical integration part of the onboarding process is performed in a strategic way, as well as how they are done in practice at the case company. We found that there are some strategies that are applied naturally; however, it is clear that there is no structured process for these strategies and there is a lot of room for improvement. Based on the results in Table 4.1, there is a somewhat successful system of peer support, either with an assigned mentor or from other team members. The fact that there is no official strategy for conducting technical integration means that there is clear room for improvement in this area at Bosch specifically. Without a robust technical integration system the time it takes until developers are confident in their work is increased and the cost of hiring new developers is subsequently increased as well.

We can also establish that the main challenge for new developers at Bosch is finding the correct information for any given situation within the companies knowledge database. This will then likely increase the likelihood of those developers needing assistance from another developer in their team. This is because whenever a new developer runs into an issue, they will have a hard time finding the information they need to resolve that issue and so must ask someone for help that either has that information or who can help them find it. This is also an extension of what was found in the related work, in which poor documentation was a major challenge for new onboarders [22]. In this case the issue is not necessarily poor

documentation but that it is difficult to find. With that said the effect of this is likely similar in that new developers are delayed in their ability to contribute effectively.

This does present the opportunity to create an LLM-based system which can alleviate this problem somewhat by assisting with the information gathering step. This way new developers only need to ask for assistance from senior developers if they have more complex problems, while many problems can be solved before there is need for that. Notably, this does not address the issue of there being no robust onboarding process at Bosch, however that is outside the scope of this thesis.

5.2 Bob Implementation (RQ2)

This section discusses the design and implementation of Bob in relation to the challenges identified in the case study and results presented in Section 4.2. The focus is on interpreting key design decisions, analyzing trade-offs, and reflecting on how different components of the system contributed to its overall performance.

A central challenge throughout development was balancing response quality with computational efficiency, particularly in terms of token usage and processing time. While the goal was to provide accurate and contextually relevant answers, practical constraints required careful consideration of how data was retrieved, processed, and utilized within the system. As a result, many design decisions were driven by the need to optimize both retrieval effectiveness and system efficiency.

The following sections discuss these aspects in more detail, focusing on data handling, system architecture, model selection, and user interaction.

5.2.1 Data Retrieval and Context Trade-offs

One of the most critical aspects of the system design was how to gather and utilize relevant data for context generation. The findings from the case study highlighted that developers primarily struggle with locating relevant documentation and understanding project context. As a result, a significant portion of the system design focused on improving data retrieval rather than solely enhancing response generation.

The use of a vector database enabled efficient semantic retrieval of both repository data and internal documentation. This approach allowed the system to provide targeted context to the language model, reducing the need to include large amounts of irrelevant information. However, this introduced a trade-off between retrieval coverage and efficiency. While more extensive data collection could increase the likelihood of retrieving relevant information, it would also lead to higher processing time and increased token usage.

To address this, a cross-referencing strategy was used to identify relevant internal documentation based on repository metadata. This approach improved retrieval precision while limiting the amount of data that needed to be processed. However, it also introduced a limitation, as some potentially relevant documentation outside the identified scope could be excluded. With greater computational resources, a broader data collection strategy, such as indexing entire documentation spaces, could further improve coverage.

The choice of vector database also influenced the development process. By selecting a solution with low setup complexity and local deployment capabilities, more effort could be

allocated to refining data processing and retrieval strategies. While alternative approaches, such as graph-based retrieval systems, may offer more advanced querying capabilities, their higher implementation complexity made them less suitable within the constraints of this project.

5.2.2 Agent-Based Architecture and System Behavior

The adoption of an agent-based architecture played a central role in shaping the system's behavior and flexibility. By decomposing the overall task into smaller, specialized components, the system was able to handle different aspects of query processing—such as query refinement, retrieval, and response generation—in a modular manner. This approach proved particularly beneficial when working with smaller language models, as it allowed each agent to operate on a limited and task-specific context.

A key observation during development was that a simple, linear processing pipeline was insufficient for handling the variability of user queries. In particular, cases where relevant information was not retrieved highlighted the need for more adaptive system behavior. To address this, additional agents and conditional logic were introduced, enabling the system to dynamically adjust its processing flow. For example, mechanisms for optimizing search queries and introducing fallback strategies improved the system's ability to recover from failed retrieval attempts.

One of the most impactful improvements was the introduction of structured input and output between agents. Early iterations relied on loosely defined prompts, which led to inconsistencies in how information was passed between components. By enforcing structured representations, the system achieved more reliable communication between agents and improved overall robustness.

Despite these improvements, the agent-based approach also introduced trade-offs. Increasing the number of agents and processing steps improved flexibility and retrieval accuracy but also led to higher token usage and longer response times. This required careful balancing between system complexity and efficiency, with some potential enhancements—such as additional validation or fact-checking steps—being excluded due to their limited benefit relative to their computational cost.

Overall, the agent-based architecture enabled a flexible and extensible system design, but its effectiveness depended on carefully managing the trade-offs between modularity, performance, and resource usage.

5.2.3 Model and Performance Trade-offs

Model selection played a significant role in balancing system performance with practical constraints such as cost, speed, and resource availability. Throughout development, different embedding and language models were considered, with the final choices reflecting a compromise between retrieval quality, response accuracy, and computational efficiency.

For embedding generation, the choice of model was influenced not only by retrieval performance but also by processing speed and suitability for local execution. While some models offered marginal improvements in retrieval quality, they were associated with significantly

slower embedding times, which negatively impacted system responsiveness during setup and iteration. As a result, the selected embedding model represented a balance between effective semantic representation and practical usability within the project constraints.

Similarly, the selection of the language model was guided by a comparison of output quality and efficiency. As multiple models produced comparable responses in terms of correctness and coherence, the final decision was based on cost and response time. This reflects an important practical consideration: in applied settings, marginal improvements in model performance may not justify increased computational cost, particularly when systems are expected to scale.

Another important factor influencing performance was the need to manage token usage effectively. Since the system relies on RAG, the amount of context provided to the language model directly impacts both cost and response time. This necessitated a design where context is selectively retrieved and constrained, rather than exhaustively included.

These observations highlight that, in practice, system performance is not solely determined by model capability, but by how well models are integrated into the overall architecture. The results suggest that careful optimization of data retrieval, context management, and system design can mitigate the limitations of smaller models, enabling efficient and effective performance without reliance on more resource-intensive alternatives.

5.3 Bob Evaluation (RQ3)

This section discusses the results from the evaluation regarding how well Bob is able to support the onboarding process at Bosch. From our thematic analysis of the user evaluations we can see that users believe Bob was good at finding and summarizing the information they were looking for. Since one of the main challenges identified in the onboarding process was being able to navigate and find relevant information in the knowledge database at Bosch this suggests that there is a positive use case for Bob at Bosch. Here we can also speculate on the possibility that this is a use case which goes beyond just technical integration during the onboarding process. The documentation database at Bosch is especially confusing for new developers, however it can also be difficult to navigate for more experienced developers, in which case Bob might be able to assist those developers as well. Our testing methodology did not set out to explore this and thus we can not make any conclusions however this is an area that could be explored further in the future.

The stand out weakness of Bob that was evident through the user evaluations was the ability for Bob to solve code tasks. While Bob was able to point to the right files the tester was supposed to work on, it was not able to point out the specific issues and code the tester needed to solve the task. This means that new developers could use Bob to find files and functions that are relevant for a given task, but they would need to solve the task on their own. When the goal is to solve code-specific tasks efficiently, one idea would be to use Bob to help the developer understand the code and find the areas that need to be worked on, and use another AI agent to assist with the code-specific parts, such as GitHub Copilot, which specializes in coding and is available for developers at Bosch.

Our user evaluations showed that developers believed it would have taken more time if they had been assigned these tasks without the use of Bob. This shows promise in the ability for Bob to expedite the technical integration part of the onboarding by contributing

meaningful work faster. It is however worth noting that our user evaluations were specifically designed with a simple-complex task assignment strategy, and thus we do not have any data on wether Bob would be helpful in the case of more complex tasks. This is something that could be explored further to see if Bob can keep up with more difficult tasks or if Bob is unable to contribute at that point. This also ties into the fact that not all tasks were solved. It was not our intention for this to be the case since we did not want the test to be too easy to solve, since then we risk having tested Bob with tasks that are not representative of real world scenarios. One way to expand the testing of Bob would be to deploy Bob to new developers at the company for use in their normal tasks, and gather data from users that way. This would provide a more accurate evaluation of how well Bob is able to assist those users, however this was both outside of our time scope and difficult due to a lack of active onboarders at the company site. This is something that would also be a good test to conduct in the future.

One of the things we wanted to look at was how well Bob could act as a mentor that new developers could get guidance from in the way they would from a senior developer. While some users said they would try using Bob before asking for help from colleagues, all users were unanimous in that Bob would not be able to replicate the assistance from more experienced team members. This means that Bob should not be used as replacement for mentors and peer assistance. The technical integration part of the onboarding process is still served best by having mentors and peers assist new developers, and then Bob can be used as an additional tool to assist in that process.

Another side of our evaluations were the internal validations that were conducted throughout our project. This was an undocumented part of our evaluations that helped guide our design choices during our implementation of Bob. Here there is a possibility to conduct more formal testing through methods such as BLEU, ROUGE, and F1 scores. This would primarily serve to measure hallucinations and instruction adherence which is important for a system to be reliable to users. This type of testing was beyond the time scope of this project, however is something that would be important to do if Bob was to be further developed and deployed at Bosch.

Overall the results show that Bob was received positively. This matches the evaluations done in the related work, in that an LLM system does show promise in improving the onboarding process. We have also further contributed to this by conducting our tests in an industrial setting which shows that this concept is not just applicable in theory, but also has potential in commercial settings. If further work is conducted to improved Bob, it could be deployed and used to assist new developers at Bosch with their technical integration. There is also a possibility to further test Bob with other developers to assess the possibility of a wider use case, in which Bob assists with formation searching and summarizing to more senior developers as well.

5.4 Threats to Validity

As in any research, there are several factors that limit the validity of this report. It is important to acknowledge those threats to add context and trust to our results, and to encourage further research to eliminate them. The following seections discuss potential threats related to generalisability, study design, evaluation, and technical constraints.

5.4.1 External Validity (Generalisability)

A primary threat to external validity arises from the limited scope of the case study. The interview study was conducted with a relatively small number of participants, all of whom were employed at the same company. As a result, the findings related to onboarding challenges may reflect the practices of Bosch, rather than being representative of software development organizations in general.

This limitation affects generalisability of both the identified problem and the proposed solution. While the system was designed to address challenges observed at Bosch, its effectiveness in other organizational contexts may vary. For example, companies with a different documentation scope may not experience the same benefits.

Additionaly the literature review used to provide a broader picture, faced other limitations. For both the literature analysis of onboarding practices and LLM techniques and evaluations, there is a vast amount of information that can be analyzed, which could not be covered within the available time. As a result, while we developed a generally solid understanding of the fields, it is far from complete. It is likely that we have missed information that could be useful to our project.

5.4.2 Internal Validity (Bias and Study design)

Internal validity may be affected by potential biases in both data collection and evaluation. The interviews conducted as part of the case study relied on self-reported experiences, which may be subject to recall bias or individual interpretation. Furthermore, the limited number of participants increases the risk that the findings are influenced by individual perspectives rather than consistent patterns.

The user evaluations of the system also present a potential source of bias. As participants may be affected by the fact that we have a personal relationship with them. Because of this, it is possible that they would be less inclined to criticize Bob, since they would not want to come off as overly negative towards the project we have been working on for the past few months. If we could have hosted our tests in a more anonymous setting, the testers might be more open about potential issues they see with Bob.

In addition, the development of the system were primarily conducted by the authors. While internal evaluation enabled rapid iteration, it also introduces the risk of confirmation bias, where design decisions may be influenced by expectations rather than objective assessment.

5.4.3 Construct Validity (Measurement and Evaluation)

Evaluation the performance of LLM-based systems presents inherent challenges, which may affect construct validity. The system was assessed using a combination of internal validation methods and qualitative user evaluations. While these approaches provide usefult insights, they do not fully capture all aspects of system performance.

For example, the evaluation of retrieval quality and response accuracy relied partly on manual assessment, which introduces subjectivity. Similarly, user evaluations were based on

perceived usefulness and correctnessm, which may vary between individuals. This makes it difficult to ensure the evaluations consistently measures the intended constructs.

Furthermore, the stochastic nature of LLMs means that system outputs may vary between runs, even for identical inputs. Although consistent results were observed during testing, it is possible that different outcomes could occur under other conditions, particularly with a larger or more diverse set of users.

5.4.4 Technical and System Limitations

The validity of the results is also influenced by technical constraints encountered during development. Due to limited computational resources, the system was implemented using smaller and more cost-efficient models, and without access to more advanced or proprietary solutions. As a result, the performance of the system may not reflect what could be achieved with more powerful models or infrastructure.

Additionally, certain design choices were influenced by practical limitations rather than purely optimal solutions. For example, data collection was restricted to a subset of available documentation which may have reduced retrieval coverage. Similarly, some advanced techniques were not implemented due to time and complexity constraints.

Finally, LLM-based systems are also prone to hallucinations. While we made efforts to make sure this issue was minimized through internal validations, it is possible that Bob will occasionally hallucinate.

5.5 Ethical Considerations

The development and deployment of systems such as Bob raise several ethical considerations regarding the role of developers and the use of AI-assisted tools in software engineering workflows.

One concern is the potential for such systems to contribute to the replacement of developers. While large language models have demonstrated increasing capabilities, the intention of this work was not to replace developers, but to support them. Bob was designed as a tool to assist developers in navigating complex project environments, rather than performing development tasks autonomously. In this sense, the system promotes a symbiotic relationship, where the developers remains responsible for decision-making while leveraging the system for improved access to information.

Another concern relates to the potential loss of developer knowledge. By relying on systems like Bob for retrieving and summarizing information, developers may become less inclined to build their own understanding of the codebase and documentation. This could lead to a form of knowledge dependency, where critical insights are mediated through the system rather than developed through direct interaction with the source material.

Overall, while systems such as Bob offer clear benefits in improving efficiency and accessibility of information, their use should be approached with consideration of these ethical implications, ensuring that they better developer expertise and responsibility rather than lessen.

Chapter 6
Conclusions

This thesis set out to investigate and support the current technical integration part of the onboarding process at Bosch, with a particular focus on how an LLM-based system can assist developers during onboarding. Through a case study of existing practices, the development of an LLM-based assistant, and an evaluation of its function in onboarding scenarios, this work aimed to assess both the potential and limitations of such systems in a corporate software engineering context.

Based on the literature review of current onboarding process, we compiled a list of common onboarding strategies and task assignment strategies. The literature guided our decisions in designing our interviews to investigate Bosch's current technical integration process (RQ1). We found that onboarding is largely informal and highly dependent on the team to which a developer is assigned. New developers are assigned tasks in a simple-complex scheme; they are supported by peers both formally and informally. This approach was generally perceived as effective, with common challenges, such as information search, niche technical issues, and development environment setup.

To address these problems, we designed and implemented an LLM-based system, Bob, intended to support developers during technical integration by aggregating and presenting relevant information in a project-based context (RQ2). Rather than attempting to replace existing practices, Bob was developed as a complementary tool, focusing on assisting users with common questions, smart documentation lookup and simpler technical problems. The system leverages LLM capabilities to reduce friction in information search and provide timely support during developer onboarding.

Lastly, the usability and effectiveness of Bob was evaluated (RQ3). Although the scope of testing was limited, the results were promising. The evaluation indicated that Bob can help onboarders locate information more quickly and assist with simpler issues that would otherwise require time-consuming searches or appealing to colleagues. At the same time, the evaluation highlighted clear limitations: Bob is not capable of replacing senior developers in mentoring roles, nor can it reliably resolve highly specific code-related problems. These findings suggest that LLM-based systems are best suited as supplementary tools rather than

standalone solutions in the technical integration process.

Overall, Bob demonstrates potential as a low-cost way to support Bosch's developer onboarding process, though the extent of its effectiveness remains uncertain. To get definitive answers, a longer case study comparing onboarding experiences with and without similar LLM-based onboarding systems has to be carried out, enabling a more precise assessment of its impact on onboarding efficiency and developer productivity.

The findings of this thesis contribute empirical evidence on the use of LLM-based workflows in technical integration. By identifying both the benefits and current shortcomings of such systems, this work can serve as a reference for future research and industrial initiatives aiming to improve technical integration through AI-based support tools. We have also contributed to Bosch with a functioning prototype that could be further developed and deployed by the company.

References

[1] sentence-transformers/all-MiniLM-L6-v2 · Hugging Face. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2, January 2024. [Online; accessed 29. Jan. 2026].

[2] sentence-transformers/all-mpnet-base-v2 · Hugging Face. https://huggingface.co/sentence-transformers/all-mpnet-base-v2, January 2024. [Online; accessed 29. Jan. 2026].

[3] Chroma. https://docs.trychroma.com/docs/overview/introduction, December 2025. [Online; accessed 4. Dec. 2025].

[4] LangChain overview - Docs by LangChain, dec 2025. [Online; accessed 4. Dec. 2025].

[5] PocketFlow. https://github.com/The-Pocket/PocketFlow, December 2025. [Online; accessed 4. Dec. 2025].

[6] What is the Model Context Protocol (MCP)? - Model Context Protocol. https://modelcontextprotocol.io/docs/getting-started/intro, 2025. [Online; accessed 23. Oct. 2025].

[7] What is Confluence Cloud? | Confluence Cloud | Atlassian Support. https://support.atlassian.com/confluence-cloud/docs/what-is-confluence-cloud, January 2026. [Online; accessed 29. Jan. 2026].

[8] Afnan Alabdulwahab, Chloe Japic, Colby Le, Disha Dubey, Disha Trivedi, John Hope, Patrick Stone, Sanjana Srivastava, Adam Tashman, and Aidong Zhang. Comparative study of large language model evaluation frameworks with a focus on nlp vs llm-as-a-judge metrics. In *2025 Systems and Information Engineering Design Symposium (SIEDS)*, pages 410–415, 2025.

[9] Erdogan Berrin Bauer, Talya N. *Organizational socialization: The effective onboarding of new employees.* American Psychological Association, 2011.

[10] Ertuğrul Bayraktar, Cihat Bora Yigit, and Pinar Boyraz. Tailoring the ai for robotics: Fine-tuning predefined deep convolutional neural network model for a narrower class of objects. In *Proceedings of the 2017 International Conference on Mechatronics Systems and Control Engineering*, ICMSCE '17, page 10–14, New York, NY, USA, 2017. Association for Computing Machinery.

[11] Jim Buchan, Stephen MacDonell, and Jennifer Yang. Effective team onboarding in agile software development: techniques and goals, 2019.

[12] Jurgen Claassen. ['Gold standard', not 'golden standard']. *Ned. Tijdschr. Geneeskd.*, 149(52):2937., December 2005.

[13] Margaret-Anne Storey Emelie, Engström and Runeson Per. *The Design Science paradigm as a Frame for Empirical Software Engineering*, pages 127–147. Springer, 2020.

[14] Stefan Feuerriegel, Jochen Hartmann, Christian Janiesch, and Patrick Zschech. Generative AI. *Bus. Inf. Syst. Eng.*, 66(1):111–126, February 2024.

[15] Michael Günther, Jackmin Ong, Isabelle Mohr, Alaeddine Abdessalem, Tanguy Abel, Mohammad Kalim Akram, Susana Guzman, Georgios Mastrapas, Saba Sturua, Bo Wang, Maximilian Werk, Nan Wang, and Han Xiao. Jina embeddings 2: 8192-token general-purpose text embeddings for long documents, 2024.

[16] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.*, 43(2), January 2025.

[17] Statista Market insights. the impact of chatgpt economics. `https://www.statista.com/forecasts/1474143/global-ai-market-size`, 2025.

[18] Andrei Cristian Ionescu, Sergey Titov, and Maliheh Izadi. *A Multi-agent Onboarding Assistant based on Large Language Models, Retrieval Augmented Generation, and Chain-of-Thought*, page 1208–1212. Association for Computing Machinery, New York, NY, USA, 2025.

[19] Per-Olov Johansson and Bengt Kriström. *Cost–Benefit Analysis*. Elements in Public Economics. Cambridge University Press, 2018.

[20] An Ju, Hitesh Sajnani, Scot Kelly, and Kim Herzig. A case study of onboarding in software teams: Tasks and strategies. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 613–623, 2021.

[21] Clare-Marie Karat. 4 chapter - a business case approach to usability cost justification for the web. In Randolph G. Bias and Deborah J. Mayhew, editors, *Cost-Justifying Usability (Second Edition)*, Interactive Technologies, pages 103–141. Morgan Kaufmann, San Francisco, second edition edition, 2005.

[22] Knud Ronau Larsen, Magnus Edvall, Truong Ho-Quang, Felix Dobslaw, and Rodi Jolak. *An LLM Assistant for Software Project Onboarding*, page 1345–1352. Association for Computing Machinery, New York, NY, USA, 2025.

[23] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021.

[24] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[25] Sabrina J. Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y. Lee, Benoît Sagot, and Samson Tan. Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp, 2021.

[26] Noah, Mayerhofer and Sandra, Nyström. Designing a Machine Learning Application to Obtain Customer Insights in the Banking Domain, 2022. Student Paper.

[27] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA, 2002. Association for Computational Linguistics.

[28] Jeffrey Rubin. *Handbook of usability testing : how to plan, design, and conduct effective tests*. Wiley, 1994.

[29] Yutaka Sasaki. The truth of the f-measure. *Teach Tutor Mater*, 01 2007.

[30] Toni Taipalus. Vector database management systems: Fundamental concepts, use-cases, and current challenges. *Cognitive Systems Research*, 85:101216, 2024.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[32] J White. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*, 2023.

Appendices

Appendix A
Onboarding Methods and Strategies

Simple-complex	Starting with simple tasks and then moving on towards more complex ones
Priority-first	Starting with the most important task regardless of difficulty
Exploration based	The developer is allowed to explore the project and find their own tasks
Mentoring	Having a dedicated mentor that assists the learning process
Peer support	Getting help from other team members
Online search	Searching for information online
Reading documentation	Reading project documentation
Pair programming	Pair programming
Self-learning	Learning about libraries, tools and techniques online resources
Knowledge database	Learning from a shared local knowledge database where thing like product information, design decisions, testing architecture and coding standards are stored.
Product overview	A presentation that shows the functionality and features of the product.
Team meeting	A meeting where team members discuss the product and what they are currently working on
AI chat tools	AskBosch for example

Appendix B
Interview Questions

- For which role were you recruited? (e.g., Junior Developer, Senior QA Engineer)
- How much experience in years did you have before beginning that role?
- How long did it take you to become familiar with the development environment setup?
- Which part of the technical integration process was the most challenging? (Environment Setup, Codebase, Design and Development Workflow)? Can you give a specific example of that?
- What one change could have made that specific part easier or faster?
- What was the most common question you or your teammates had to ask a senior developer or search for in the internal documentation?
- What was the first task you were assigned? (e.g., debugging, error analysis, testing, adding new features)
- How well did the technical integration process prepare you to complete that task?

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2026-03-16

EXAMENSARBETE Creating an Onboarding Tool for Technical Integration Using an LLM Approach
STUDENTER Axel Nåsell, Harry Norrman
HANDLEDARE Sule Tekkesinoglu (LTH)
EXAMINATOR Elizabeth Bjarnason (LTH)

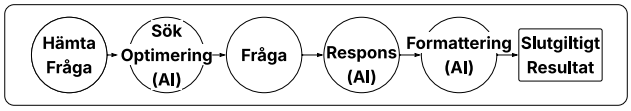
En effektivare process för introduktion av nya programvaruingenjörer med hjälp av en AI-assistent

POPULÄRVETENSKAPLIG SAMMANFATTNING **Axel Nåsell, Harry Norrman**

När teknik-företag växer snabbt blir det allt viktigare att hjälpa nyanställda att komma igång med sina tekniska verktyg och arbetssätt. I det här arbetet undersöks vilka hinder som finns i dagens introduktion av nya programvaruingenjörer. Som ett resultat har en AI-assistent tagits fram för att fungera som ett digitalt stöd när nya medarbetare lär känna programvaran och verktygen för den IT produkt eller tjänst som de ska utveckla

Introduktion av nya programvaruingenjörer är ett tidskrävande moment för utveckling av digitala system som involverar många tidskrävande moment som behöver genomföras innan nya programmerare kan göra meningsfulla bidrag till ett projekt. Ett av de viktigaste stegen i denna process är den tekniska integrationen, vilket syftar på inlärning av verktyg och arbetssätt. För att underlätta denna process har vi i detta examensarbete tagit fram en AI-assistent till ett företag. Denna assistent kan hitta och sammanfatta stora mängder intern dokumentation för att underlätta denna inlärningsprocess.

Efter en analys av den nuvarande tekniska integrationsprocessen på företaget byggde vi assistenten med hjälp av en teknik som kallas Retrieval Augemented Generation (RAG). Detta är en populär AI-teknik som går ut på att hämta information i realtid när en användare ställer en fråga till en AI-assistent. Denna information kan sedan matas in som kontext till den originella frågan för att på så sätt användas av assistenten i sitt svar. Vår slutliga assistent baserades på en flödesmodell, där AI-baserade noder och systemkomponenter successivt skickar data till varandra för att på bästa sätta formatera data för både interna komponenter och den slutliga utmatningen som användaren ser. Denna modell kan ses i figuren nedan.



Efter utvärdering av assistenten kunde vi se att programmerare var positivt inställda till att använda assistenten för att hitta information och de rapporterade att de löste sina uppgifter snabbare. Däremot så påpekades det att assistenten inte kunde ersätta assistans från erfarna programmerare men att det kunde vara värdefullt för enklare uppgifter.

Detta arbete har undersökt möjligheter med AI-assisterad introduktion för nya programvaruingejörer och hur den tekniska integrationen kan förbättras. Vi ser att en AI-assistent kan vara hjälpsam och det finns god potential att ytterligare förbättra processen.