Institute of Software Engineering

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Analyzing Software Architectures of Open-Source AI-based Software

Max Kästner

Course of Study:	Software Engineering
Examiner:	Prof. Dr. Stefan Wagner
Supervisor:	Markus Haug, M.Sc.
Commenced:	October 2, 2023
Completed:	April 2, 2024

Abstract

Recently, software systems that include artificial intelligence (AI) components (also called AI-based software systems) have become increasingly popular. However, the integration of AI into software systems brings new challenges compared to traditional software systems. In particular, the software architecture of AI-based software systems needs more research. In this thesis, we investigate the software architecture of open-source AI-based software applications, with an explicit focus on the design patterns and architectural tactics used in the architectures. For this purpose, we have mined open-source AI-based software systems from GitHub to analyze the software architecture of the software systems. To identify the design patterns, we use a design pattern catalog specifically created for AI-based software systems. We analyzed a total of 3 open-source AI-based software systems and were able to identify a total of 19 unique design patterns, with a balanced number of found traditional adapted design patterns and new AI-specific design patterns. We could see that design patterns from the categories *Architecture* and *Deployment* were more frequently identified in the systems than in the other categories *Implementation*, *Process*, *Security & Safety* and *Testing & Quality Assurance*. It was noticeable that many design patterns in the *Architecture* category were adapted from traditional design patterns, while several new AI-specific design patterns were found for *Deployment*, solving the problem of how to deploy the machine learning (ML) models. While a lot of new AI-specific design patterns for *Security & Safety* have been established, we could only identify very few. In addition, testing AI components is very unpopular in the analyzed systems and difficult as the correctness of an ML model is hard to test.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AI artificial intelligence. 3, 15

GoF Gang of Four. 17

LLM Large Language Model. 28

ML machine learning. 3, 19

OSS open-source software. 17

1 Introduction

Software systems based on artificial intelligence (AI) have gained considerable interest in a wide range of fields. More and more software systems contain AI components to enable various functions, such as face recognition in photo management applications or conversations with chat bots. Such systems are AI-based systems, which contain one or more AI software components [MBF+22]. One particularly underresearched aspect of AI-based software systems is their software architecture. On the other hand, existing open-source applications are investigating how to integrate AI-based functionality and new open-source applications are built on AI from the start. Without any guidance, the developers working on these applications are forced to design the software architecture by trial and error.

Much of the literature about the use of AI in software systems is primarily based on external research through interviews or surveys with industrial partners [IY19; LOX21; NZLK22; SV21; VB19] or research within a company and gaining comprehensive access [BME19; HBB+18; HCHV21; RRK+19]. There is far too little work that actually looks at open-source AI-based software systems themselves. With this motivation, we analyzed open-source AI-based software systems on their software architecture, focusing on the design patterns and architectural tactics used in the software systems to find out what the software architecture looks like in the real world.

Our goal is to determine from open-source AI-based software systems what the software architecture of AI-based software systems looks like, what design patterns and architectural tactics are used to implement them, and how the design patterns in AI-based software systems differ from those in traditional systems. Therefore, this study will try to answer the following research questions:

- **RQ1:** How are design patterns and architectural tactics applied in AI-based software systems?
- **RQ2:** How do design patterns in AI-based software systems differ from those in traditional software systems?

To answer this research questions, we first collected a catalog of AI-based software systems from GitHub to have a broad selection of AI-based software systems. Then, we analyzed the software architecture of some software systems and specifically identified design patterns using a design pattern catalog specifically for AI-based software systems.

This thesis is organized as follows. Chapter 2 explains the necessary background knowledge. Chapter 3 provides related work that is similar to the work of this thesis. Next, Chapter 4 describes the methods used in this thesis, divided into the repository collection in Section 4.1 and the repository analysis in Section 4.2. Chapter 5 shows the results of the repository collection and analysis according to our research questions. Chapter 6 discusses the results of the study and threats to validity. Finally, Chapter 7 concludes with a summary of the thesis.

2 Background

This chapter introduces the basics of software architecture, design patterns, and AI-based software systems.

2.1 Software Architecture

Software architecture refers to the fundamental structure of a system that encapsulates its overall design choices [Gar00]. It outlines how the system is assembled from interacting components, identifies primary interaction pathways, and defines essential properties of these components. An architectural description provides sufficient detail for high-level analysis and evaluation, enabling a comprehensive understanding and evaluation of the system design.

Software architecture documentation is a critical component of the architecture design process, helping architects identify and document necessary design decisions [DLT+14]. It provides stakeholders, such as architects, with accessible and unambiguous information essential to their roles [CGL+03]. In the context of open-source software (OSS) adaption, comprehensive software architecture documents positively impact the degree and cost of OSS adaption [AW07]. They help users understand the transition from design issues to architectural solutions, understand the system model, its purpose, architectural requirements, and more [DLT+14]. However, many OSS do not document their software architecture [DLT+14].

2.2 Design Patterns

Design patterns are recurring structures of design and architecture [Mar00]. They are proven and established solutions to a common design problem. Like many practices in software engineering, design patterns have been refined based on the collective experience of software developers. Typically, a pattern contains at least four essential components: the pattern name, the problem statement, the proposed solution, and the consequences [HHB23]. As such, it serves as a collection of principles rather than a rigid set of step-by-step instructions. As a result, the specific execution details may vary between different applications of the pattern.

The Gang of Four (GoF) book by Gamma et al. [GHJV95] marked the first effort to create a standard catalog of 23 patterns. A design pattern catalog is a structured list of design patterns that address specific problems and propose solutions. The goal is to make design patterns easily accessible. Several domain-specific catalogs or catalogs for specific application areas have been established, such as for Microservices [TLP18] or Internet of Things [WOH+20].

2.3 AI-based Systems

AI-based systems are systems which contains at least one AI component [MBF+22]. According to the definition of Martínez-Fernández et al., these systems analyze their environment and perform actions, aiming to acquire intelligent behavior through learning. AI-based systems can exist purely as software that operates in the digital realm, such as voice assistants, image analysis software, search engines, and speech and facial recognition systems [MBF+22]. Alternatively, AI can be integrated into hardware devices such as advanced robots, autonomous cars, drones, and Internet of Things applications.

3 Related Work

In this chapter, we provide an overview of related work that deals with AI-based systems, including their software architecture. However, a large part of the related work deals with machine learning (ML) systems.

In 2022, Martínez-Fernández et al. [MBF+22] conducted a systematic mapping study to comprehend the landscape of Software Engineering approaches for AI-based systems. They examined 248 studies published from January 2010 to March 2020, revealing Software Engineering for AI-based systems as an emerging research field, with more than two-thirds of the studies published after 2018. Most studies focus on making AI-based systems reliable and secure. Their findings are relevant to researchers attempting to understand the current state of the art and identify areas for further research. They also provide practitioners insights into Software Engineering approaches and challenges specific to AI-based systems, while helping educators bridge the gap between Software Engineering and AI in their curricula. While they found that there are many contributions to software testing and software quality for AI-based systems, software construction, software requirements, and especially software maintenance seem to be less represented and offer a lot of potential for researchers.

While numerous systematic reviews have explored general design patterns [BRG17], only a limited number of studies have focused on collecting design patterns specifically targeted to AI-based systems. In 2019, Washizaki et al. [WUKG19] conducted a systematic literature review to collect, classify and discuss Software Engineering design patterns for ML techniques, which was the first survey and comprehensive literature review on ML architecture and design patterns. Later in 2022, Washizaki et al. [WKG+22] published 15 identified design patterns for ML applications from a multivocal literature review. Heiland et al. [HHB23] presented an overview of design patterns for AI-based systems. To do this, they conducted a multivocal literature review to collect design patterns with AI-based systems, so that they could include gray literature, which benefits from being close to a practical context [GFM16], since there are not many scientific publications on this topic. From 51 resources, they extracted 70 unique patterns used for AI-based systems, of which 34 were new AI-specific patterns and 36 were traditional patterns adapted to AI-based design patterns. They made these design patterns available and categorized them in a web-based pattern repository[1] to make the patterns searchable and easy to find for researchers and practitioners. For our study, we want to use the design pattern catalog by Heiland et al. [HHB23] as a basis for our study to identify design patterns in AI-based systems.

Nahar et al. [NZL+23] identified in their study open-source ML products from GitHub repositories to facilate further research and education with ML products. ML products are described as software products for end-users that contain ML components, and are distinct from other ML-related software projects and artifacts (e.g. exploratory notebooks or course homework). The research revealed a

[1] https://swe4ai.github.io/ai-patterns/

diversity of development practices and architectural decisions in ML models, providing opportunities for future innovation, but also revealed a concerning lack of industry best practices such as model testing and pipeline automation in open-source ML products, prompting further investigation into their impact on product development and end-user experience.

Gonzalez et al. [GZN20] conducted a comprehensive empirical analysis of AI and ML repositories hosted on GitHub with the goal of characterizing this specific community of developers and identifying unique characteristics, development patterns, and trends within the AI and ML domain. They compared three types of repositories, which were AI and ML Tools (frameworks & libraries), Applied AI and ML (repositories using AI and ML) and repositories which were unrelated to AI and ML. It aimed to map the timeline of the AI & ML "boom", examine ownership structures, understand popularity metrics and dominant programming languages, and delve into the collaboration dynamics and team autonomy within these repositories. These aspects were examined as critical factors influencing productivity in AI & ML software development. Compared to their analysis, our study focuses on the type of software which they call "Applied AI and ML" and only concerns the software architecture of this software.

Serban and Visser [SV21] aimed to explore how software systems can be designed or redesigned to effectively integrate ML components. They used a mixed-methods approach that included a systematic literature review, semi-structured interviews with industry practitioners, and a survey. This comprehensive method enabled them to identify and validate 20 specific challenges and solutions related to architecting systems with ML components.

4 Methodology

This chapter focuses on the study process and the methodology used. Figure 4.1 provides an overview of the major stages of our research process. To answer our research questions and gain insight into the software architecture of OSS, the first step was to obtain a list of projects that are AI-based software systems to analyze. We decided to search the GitHub platform for a possible list of suitable projects, although it is not trivial how to search GitHub based on certain parameters. More details are explained in Section 4.1.1. After understanding how to search GitHub, it was a matter of determining the parameters to search and getting a list of candidates. In Section 4.1.2 we address what parameters we chose to obtain a list of AI-based systems from GitHub. Since not all of the criteria for the projects could be checked fully automatically, it is also a matter of narrowing down the list of possible candidates to allow for a manual inspection of the repositories to determine whether the project should be considered for analysis. Once we have a list of AI-based software systems, the next step is to analyze these projects according to their architectural tactics and design patterns in order to draw conclusions and answer our research questions. This process is described in Section 4.2.

4.1 Repository Collection

This section describes the process of how we collect our repositories. There are different ways to collect repositories with certain properties from GitHub, which we explain in Section 4.1.1. The actual process, based on which properties we have collected the repositories, is explained in Section 4.1.2.

4.1.1 Mining Platform

Since our main motivation for this study is to analyze the architecture of AI-based software systems, OSS are particularly interesting for us to learn and gain insight into how developers design the architecture of AI-based systems. In a software repository mining study [KGB+14], there are

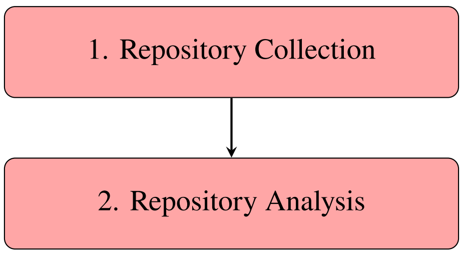

Figure 4.1: Methodology Process

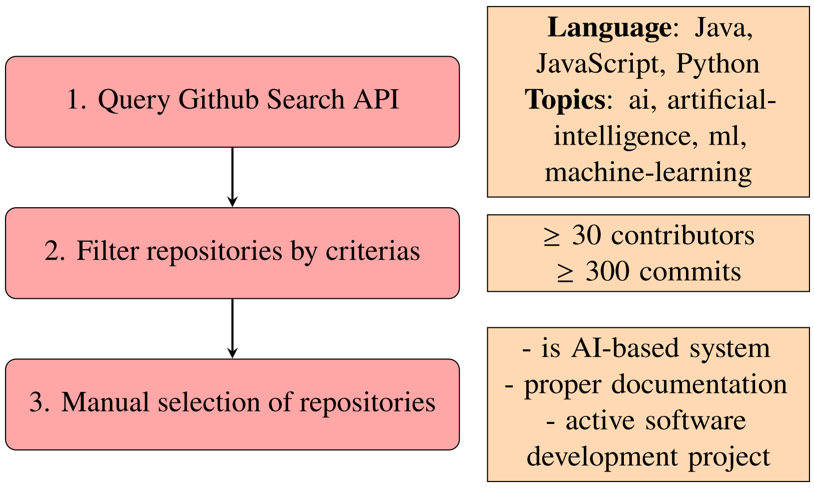
1. Query Github Search API
2. Filter repositories by criterias
3. Manual selection of repositories
Language: Java, JavaScript, Python
Topics: ai, artificial-intelligence, ml, machine-learning
≥ 30 contributors
≥ 300 commits
- is AI-based system
- proper documentation
- active software development project

Figure 4.2: Repository Selection

several ways to obtain the objects. With over 420 million git repositories[1], GitHub[2] becomes one of the most important sources of software artifacts on the web. GitHub is a collaborative code hosting site with social features such as project tracking and user following. To mine data from GitHub, there are several ways to extract data from GitHub. Specifically, GitHub provides an official REST API[3] that allows access to project information via HTTP requests. Given the considerable interest in GitHub for research purposes [GTS+22; GZN20; NMY+20; NZL+23], additional platforms and datasets, such as GitHub Archive[4] and GHTorrent [GS12], have been created to collect data from this platform and make them available.

To access data using the REST API, requests must be sent to `https://api.github.com/` with specifications for the desired data. For example, querying `https://api.github.com/search/repositories?q=topic:ai`, which can be specifically named the Search API, allows you to search for repositories tagged with the topic 'ai'. However, the Search API is limited to 30 authenticated and 10 unauthenticated requests per minute. The GitHub Search API only returns the first 1000 results per search, which is another limitation worth mentioning.

4.1.2 Repository Collection Process

The methodology used to collect AI-based repositories from GitHub involves a multi-step process as shown in Figure 4.2. This structured approach aims to ensure the selection of relevant, actively maintained, and well-documented repositories, while excluding non-AI systems, frameworks, tools, or inactive projects.

1. Query GitHub Search API: For the repository collection from GitHub, we created a Python script which queries the GitHub Search API to search for repositories based on the following criteria:

[1] `https://github.com/about`
[2] GitHub, `https://github.com`
[3] GitHub REST API, `https://docs.github.com/en/rest?apiVersion=2022-11-28`
[4] `https://www.gharchive.org/`

Language: Java, JavaScript, Python

Topics: ai, artificial-intelligence, ml, machine-learning

Forks: ≥ 20

Last update: $\geq$ 2023-07-01

The languages were selected based on their prevalence and widespread use in AI and ML projects [GZN20]. These topics were chosen to filter repositories specifically related to AI and ML to ensure relevance to the research focus. In our selection process, repositories with at least 20 forks are chosen for several reasons. First and foremost, this threshold serves as a measure of community interest and engagement in the projects. It also takes into account the limitations of GitHub's Search API, which limits results to the first 1000 repositories for a given query; with a minimum of 20 forks, we were below the 1000 repository limit per query, thus reducing the overall list of repositories while filtering out repositories with less community interest. Furthermore, the 20 forks criterion is consistent with GitHub's dominant "fork & pull" model [JLH+17]. Repositories with a higher number of forks generally indicate greater community interest and potential collaboration, making them more likely to provide valuable insights and contributions to the AI landscape. Since we require a minimum of 30 contributors per repository in step 2, 20 forks seems appropriate. Finally, we limit the search to repositories that have been updated by 2023-07-01 to ensure actively maintained repositories. A pseudocode of our Python script to query the GitHub Search API is shown in algorithm 4.1.

Algorithm 4.1 Selecting repositories from GitHub Search API

```
topics = ["ai", "artificial-intelligence", "ml", "machine-learning"]
languages = ["java", "js", "python"]
min_forks = 20
last_push = "2023-07-01"

per_page = 100 # 100 is maximal items per page

list = []

for topic in topics:
    for language in languages:
        last_page = 1
        current_page = 1
        # iterate over all pages for the request
        while current_page <= last_page:
            result = request github search api with parameters topic, language, min_forks, last_push, current_page
            for repo in result["items"]:
                if repo not in list:
                    fetch commits and contributors amount from GitHub repository page
                    add repo to list
            total_results = result["total_count"]
            last_page = roundup(total_results / per_page)
            current_page += 1
```

2. Filter repositories by criteria: Repositories that do not have at least 30 contributors are discarded in order to preserve repositories that have an active community for developing OSS. We also filter out repositories that do not have at least 300 commits to filter out inactive repositories, since the activity on GitHub is mostly reflected in commits [KGB+14].

3. Manual selection of repositories: As a final step, we performed a manual selection of existing repositories. We started by identifying repositories similar to the work of Gonzalez et al. [GZN20]. They identified AI & ML Tools and Applied AI & ML Applications, while the latter are the AI-based systems we are interested in. These AI & ML Tools repositories typically include frameworks, libraries, and related components in the field of AI and ML. For all other repositories that we could not assign to AI-based systems or tools for AI-based systems, we specified the category "Others", which are repositories that do not contain AI-based elements. This category included various materials such as code snippets, examples, lectures, or repositories that lack documentation in either English or German.

To categorize each repository, we began the process by accessing the GitHub repository page of each individual repository in our scope. First, we examined the repository's description on GitHub, focusing on keywords such as "framework" "library", "tutorial", or similar terms. This initial scan helps us to pre-categorize repositories as either AI tools or non-AI-based systems based on the information provided in their descriptions. In cases where the category remains ambiguous or unclear from the repository description, we looked further into the README file or linked documentation sites associated with the repository. These sources often provide more detailed information that can help clarify the repository's intended category. In case there was no such documentation, the repository receives the category non-AI based system. If further clarification was needed, we took a close look at the dependency files, such as *requirements.txt*, *pom.xml*, or *package.json*. These files typically contain essential information about the dependencies and functionality used within the repository. Analyzing these files provided additional context that helped us make a more informed decision about the appropriate category for the repository.

As a final task in the manual selection process, we categorized the repositories that we classified as AI-based systems into three levels based on the quality of architecture documentation in their project documentation: contains detailed architecture documentation, basic architecture documentation, and quasi-nonexistent architecture documentation. This will also allow us to create a priority list of projects to be analyzed later.

4.2 Repository Analysis

For the analysis process, we used the repositories mined from the repository collection section, which we had already categorized into three levels based on their architectural documentation quality. Based on this categorization, we created a ranking list by selecting the repositories with the best architecture documentation level. The repositories were then sorted by popularity, which we determined based on the sum of the z-scores of GitHub stars and forks, as other studies [ABBS15; APS16; KS20] have done.

The repository analysis is a comprehensive investigation of the selected repositories, with a particular focus on software architecture, design patterns, and architectural tactics. To identify design patterns, we used the design pattern catalog for AI-based systems by Heiland et al. [HHB23], which contains

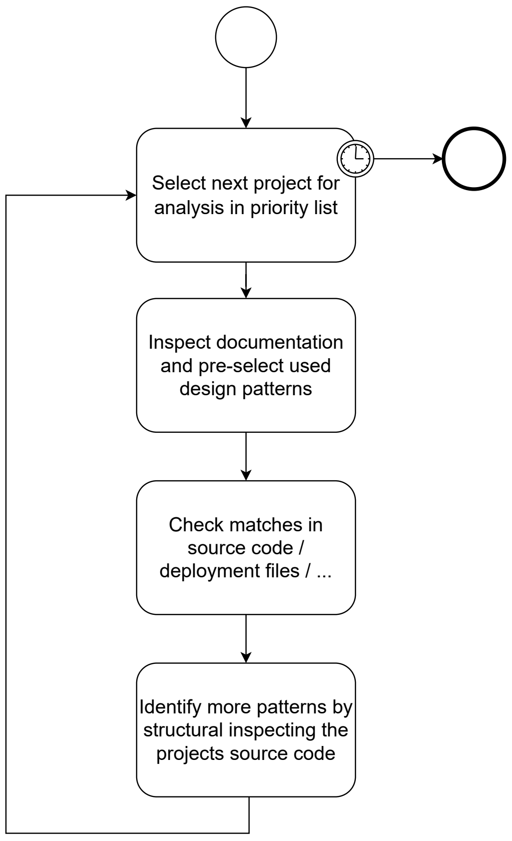
Select next project for analysis in priority list
Inspect documentation and pre-select used design patterns
Check matches in source code / deployment files / ...
Identify more patterns by structural inspecting the projects source code

Figure 4.3: Repository Analysis Process

70 unique design patterns, both adapted traditional and new AI-specific design patterns. They grouped these 70 design patterns into 7 categories: *Architecture*, *Deployment*, *Implementation*, *Security & Safety*, *Process*, *Testing & Quality Assurance* and *Topology*. A distribution of these design patterns by category is shown in Figure 4.4. Excluding the Topology category which is a subset of the Architecture category, only two design patterns appear in two different categories. This offers us a good opportunity to identify the design patterns in these groupings.

We performed a sequential analysis process, where one project from our sorted list was fully analyzed before proceeding to the next project. Each individual project was analyzed according to the following steps: First, we looked at the documentation of the project to pre-select existing architectural tactics and design patterns that might be used. After finding possible design patterns, the task was to find these described design patterns as evidence in the source code or deployment files to prove their existence. Once we had found design patterns from the documentation as possible design patterns, we were of course also on the lookout for design patterns that were not noted or made visible in the documentation. To do this, we went through the source code, deployment files (which for some projects could also have been external repositories in the same GitHub organization) or other related sources in the repositories in a structured way. By the categorizing of the design pattern catalog, we were always able to pay special attention to the specific characteristics of each category to identify design patterns. Due to the externally imposed time limit of the study, we set a strict deadline for the analysis phase. The described process is also shown in Figure 4.3.

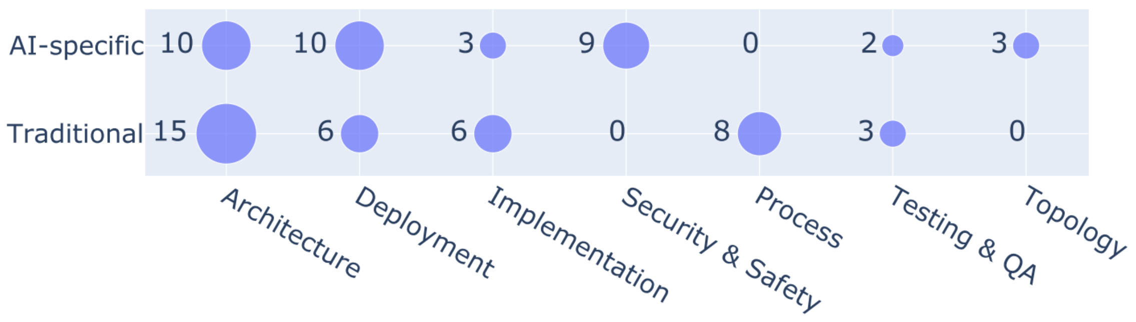

Figure 4.4: Comparison of the number of adapted traditional (36) and new AI-specific (34) patterns [HHB23]

Although there are a few tools [AZ11; GA08] to identify design patterns in software projects, we decided to carry out the identification manually. These tools are mostly based on the GoF [GHJV95] catalog and also have problems finding all design patterns or give many false positives [AZ11]. Since the catalog from Heiland et al. [HHB23] provides us with many new design patterns that do not appear in the GoF catalog, it would not be worthwhile to use or adapt such tools for this study.

5 Results

In this chapter, we present the results of the repository collection and the results of our research questions, which include the architectural tactics and design patterns found in the analyzed repositories, and we look at the design patterns used, distinguishing whether the design patterns are traditionally adapted design patterns or newly developed AI-specific design patterns.

5.1 Repository Collection

We performed all three steps mentioned in Section 4.1 to collect AI-based repositories for further analysis.

The request to the Github Search API with our specified parameters returned a total of 1466 unique repositories. After filtering out repositories that do not have at least 30 contributors and 300 commits, the number of repositories drops from 1466 to 311 remaining repositories. In the third step, we manually inspected the remaining repositories and categorized them as shown in Table 5.1. Of the 311 repositories, we identified only 37 as AI-based systems.

After closer inspection of the projects documentation, we classified the projects architecture documentation into 3 levels: contains detailed architecture documentation (+), contains basic architecture documentation (/), and contains quasi-nonexistent architecture documentation (-). Out of our 37 repositories, we identify only 2 projects documentation as detailed architecture documentation, 6 projects with basic architecture documentation, and the remaining 29 projects with quasi-nonexistent architecture documentation. We have listed the repositories with at least basic architectural documentation in Table 5.2, sorted first by the level of architectural documentation, then by the z-score of the sum of forks and stars.

Category	**Number of Repositories**	**Percentage (%)**
AI-based System	37	11.9
Library	154	49.5
Framework	53	17.0
Tool	21	6.8
Others	46	14.8
Total	**311**	**100**

Table 5.1: Distribution of categories of filtered repositories after manual selection

GitHub Repository	Architectural Documentation Level	Programming Language
https://github.com/LAION-AI/Open-Assistant	+	Python
https://github.com/LibrePhotos/librephotos	+	Python
https://github.com/AntonOsika/gpt-engineer	/	Python
https://github.com/Pythagora-io/gpt-pilot	/	Python
https://github.com/Cloud-CV/EvalAI	/	Python
https://github.com/aws-solutions/qnabot-on-aws	/	JavaScript
https://github.com/casibase/casibase	/	JavaScript
https://github.com/Josh-XT/AGiXT	/	Python

Table 5.2: AI-based System Repositories with good or basic architectural documentation; sorted by architectural documentation level, then by z-score of sum of stars and forks

5.2 RQ1: How are design patterns and architectural tactics applied in AI-based software systems?

This research question addresses design patterns and architectural tactics used in open-source AI-based systems. We have taken the repositories already mined in Section 5.1 as a basis for our analysis. We analyzed these repositories one by one, but due to time and scope constraints, we could only analyze three projects. Accordingly, we analyzed the first three projects from our sorted Table 5.2, which are Open-Assistant, LibrePhotos and GPT-Engineer:

Open-Assistant: Open-Assistant is a chat-based assistant with the main goal to make a Large Language Model (LLM) that can run on a single high-end consumer GPU. The application consists of several microservices that perform different tasks, a frontend, a backend, and additional inference services such as several inference workers, an additional security service for the inference tasks, and an inference server itself that distributes the inference tasks to the workers.

LibrePhotos: LibrePhotos is a self-hosted alternative for hosting all kinds of photos. It offers ML features such as face recognition in images, caption generation or the semantic search for objects in images. Most of the ML tasks are separated into their own small backends for processing the tasks.

GPT-Engineer: GPT-Engineer is a project that uses a LLM (such as GPT-4) to automate software engineering processes by asking for further specification of the software to be built in a conversation. The application is controlled via a command line interface.

With these three applications we have a diverse set of applications using AI. With Open-Assistant and LibrePhotos we have two web-based applications and with GPT-Engineer we have a command line based application. Open-Assistant has deployed all its services in extra containerized services, while LibrePhotos has the frontend and the backend as containerized services, where the backend has started several small backends for smaller services as subprocesses. GPT-Engineer is just an application that is started via the terminal and uses for example the GPT-4 model[1] via the OpenAI API.

We analyzed the architecture of the three projects, in particular identifying the design patterns and architectural tactics used.

5.2.1 Design Patterns

The design patterns we were able to identify from these three projects are shown in Table 5.3. As mentioned in Section 4.2, before analyzing the project itself, we examined the documentation for possible design patterns used, where we could only pre-select some design patterns used in Open-Assistant, while we could not make any assumptions about the pre-selection of design patterns used in the documentation of LibrePhotos and GPT-Eningeer based on the documentation. We found a total of 19 unique design patterns in the three projects. It can be seen that we found relatively the same number of design patterns in Open-Assistant and LibrePhotos with 13 and 12 identified design patterns, whereby the intersection of equally found design patterns is also quite high, while we only found 3 unique design patterns in the GPT-Engineer project, which is probably also related to the way AI was used in the projects. This is an average of ≈ 9 design patterns per project.

Figure 5.1 shows the number of design patterns found per category in comparison with the number of design patterns per category of the design pattern catalog by Heiland et al., with the difference that we have omitted the *Topology* category due to duplicates and only include the duplicates of *Architecture* and *Deployment* in *Architecture* in our statistics. We can see that the number of found design patterns from the *Architecture* and *Deployment* categories is significantly higher in absolute terms and also relative to the amount of design patterns in the category of the catalog than the other categories, with at least more than 30% occurrence. If we look again at the design patterns found in Table 5.3, we can see that in the categories *Implementation*, *Security & Safety*, *Process* and *Testing & Quality Assurance*, the design patterns only occur in one of the three projects, with the exception of one design pattern.

5.2.2 Architectural characteristics

During the systematic analysis of the software projects, in addition to the identification of design patterns, we also noticed special architectural tactics and other characteristics, both positive and negative.

In general, we saw that the AI components were largely encapsulated from the actual logic of the application, mostly by their own backends, which were requested via HTTP and then returned the result. With Open-Assistant in particular, we saw that they used extra worker nodes to process their AI tasks, which, as soon as they had nothing more to do, repeatedly processed new tasks from a queue (if tasks were available). These workers were deployed using Docker containers and could have been scaled quickly as required. In general, we also saw that for the two web-based projects, Docker images were built for each service. While LibrePhotos had done it rather strangely, because

[1] `https://platform.openai.com/docs/models/gpt-4-and-gpt-4-turbo`

Pattern	Open-Assistant	LibrePhotos	GPT-Engineer
Architecture			
Client-Server	x	x	
Model-View-Controller (MVC)		x	
Multi-Layer Pattern	x	x	
AI Pipelines	x		
Distinguish Business Logic from ML Model	x	x	
Synchronous pattern		x	x
Asynchronous pattern	x	x	
Parameter-Server Abstraction	x		
Deployment			
Model Embedded in Application	x	x	
Dedicated Model API		x	x
Web single pattern	x	x	
Model-in-image pattern	x		
Model-load pattern		x	
Implementation			
Strategy Pattern		x	
Security & Safety			
Secure Virtual Premise (SVP)	x		
Delegation of Safety Responsibility	x		
Process			
Infrastructure as Code	x	x	
Internal Feedback	x		
Testing & Quality Assurance			
End-To-End Tests			x
Total found Design Patterns	**13**	**12**	**3**

Table 5.3: Found Design Patterns

they had two Docker images for their project, one for the frontend and one image for the backend, which had started multiple backends for extra AI components as a sub-process in the Docker container, which would not make it possible to scale individually required AI component backends. On the other hand, Open-Assistant had the advantage that they had deployed their extra services and worker nodes as individual Docker containers, which makes scaling possible here. Both projects have provided code to run the projects in both a development and production environment, mostly using Docker Compose files.

In the *Security & Safety* area, we have only seen something about this in Open-Assistant. Open-Assistant has an additional security service for checking user prompts, which is called before a prediction for this prompt is created. This safety service checks whether the input violates any rules and evaluates this input. The input is only processed further if the evaluation does not violate the rules.

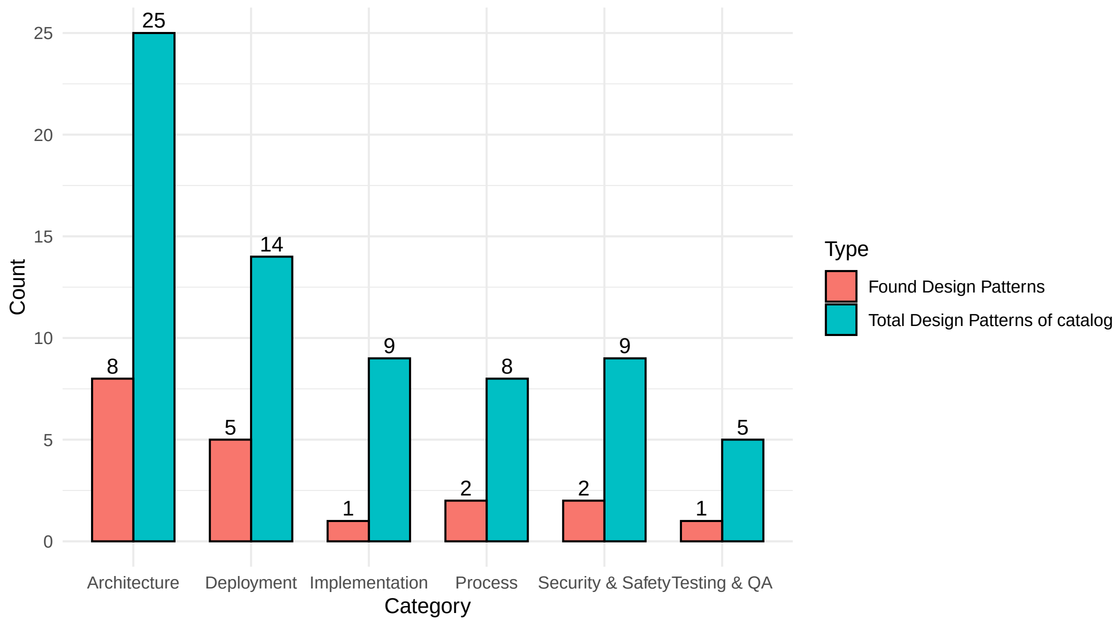

Figure 5.1: Comparison of the number of found design patterns per category

As mentioned earlier, we could only find limited occurrences in the *Process* area because they need to be documented, otherwise we would not know what the development team's process was because we were not part of it. As mentioned above, we could see that deployment was often available via code, often via Docker Compose files. For Open-Assistant and LibrePhotos, there were actions available for automated build and partial deployment via GitHub Actions, but in our opinion they did not meet the requirements for the Continuous Integration and Deployment design pattern, as they lacked either automated testing or manual deployment to production.

It was very noticeable that while we saw tests for the logic of the applications, specific tests for the AI components were missing. As in the *Process* area, we would only have been able to find out whether the applications were tested manually if this had been properly documented, which we could not find in any of the project documentation. The testing area in the analyzed systems for AI components was therefore very poor.

We also noticed that although we have sorted our projects to be analyzed by architectural documentation, the documentation did not give any reasons why the architecture of this software looks the way it does.

5.3 RQ2: How do design patterns in AI-based software systems differ from those in traditional software systems?

This research question addresses the differences between design patterns of AI-based software systems and those of traditional software systems. To address this research question, we distinguish the identified design patterns from our three projects into traditional adapted design patterns and new AI-specific design patterns as categorized by the catalog by Heiland et al. [HHB23].

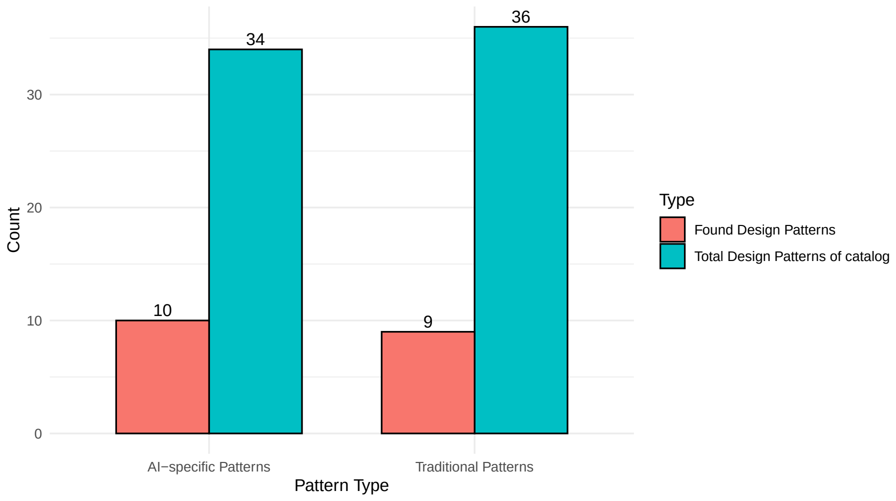

Figure 5.2: Comparison of the number of found adapted traditional and new AI-specific patterns

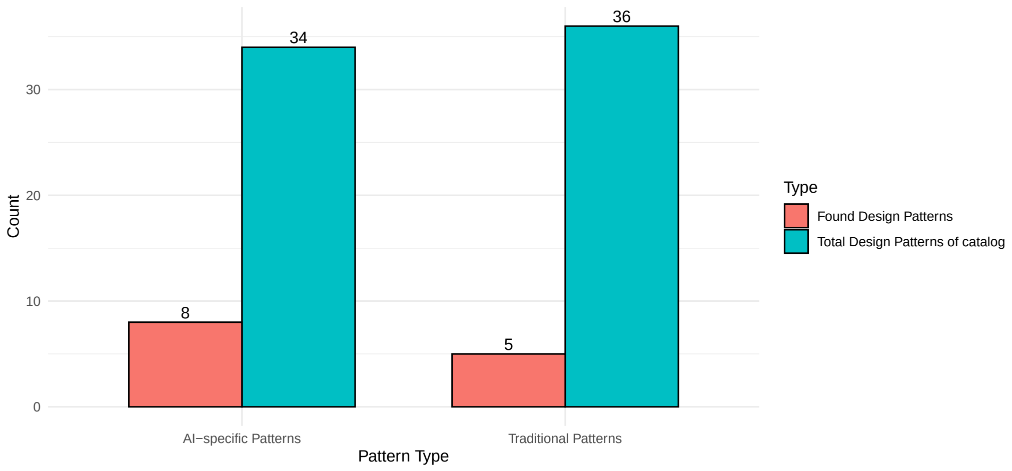

Figure 5.3: Open-Assistant: Comparison of the number of found adapted traditional and new AI-specific patterns

Of the 19 design patterns found in the three projects which were analyzed, 10 are new AI-specific design patterns and 9 are traditional adapted design patterns, as shown in Figure 5.2.

Looking at each project individually, only the Open-Assistant project used more AI-specific design patterns than traditonal adapted design patterns, as shown in Figure 5.3. LibrePhotos used slightly more traditional design patterns than AI-specific design patterns, as shown in Figure 5.4. GPT-Engineer also used one more traditionally adapted design pattern than AI-specific patterns, but GPT-Engineer already found very few design patterns, as shown in Figure 5.5.

We also have the results for the design patterns per category separately for AI-specific and adapted traditional patterns. Figure 5.6 shows the comparison of the number of new AI-specific design patterns found compared to the new AI-specific design patterns per category in the pattern catalog, and Figure 5.7 shows the comparison of the number of adapted traditional design patterns found compared to the total number of adapted traditional design patterns per category in the catalog.

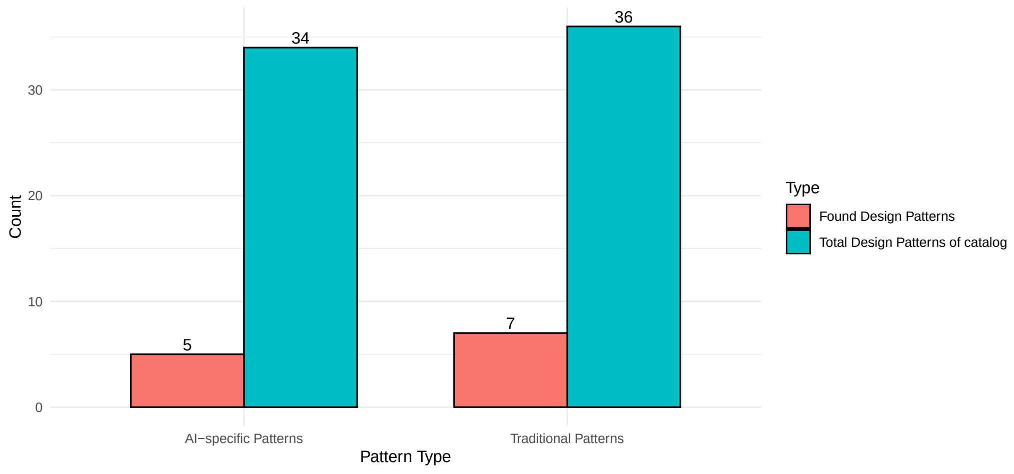

Figure 5.4: LibrePhotos: Comparison of the number of found adapted traditional and new AI-specific patterns

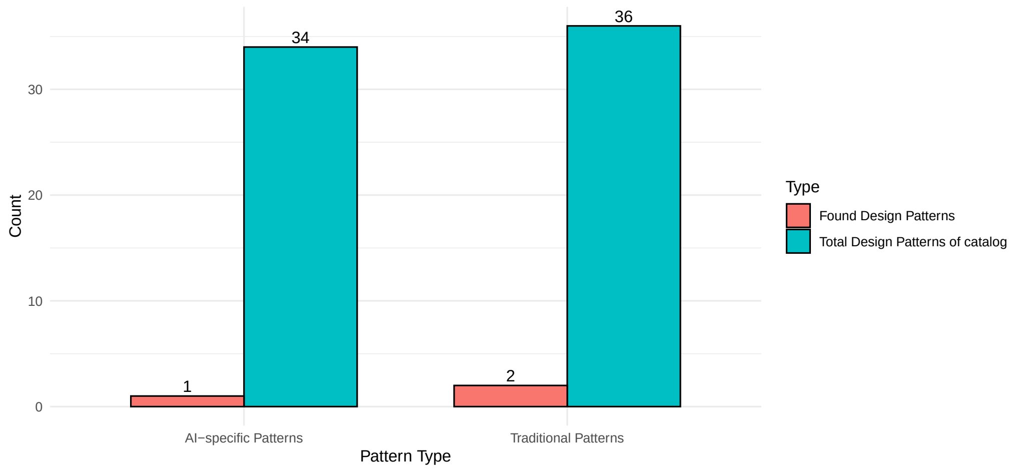

Figure 5.5: GPT-Engineer: Comparison of the number of found adapted traditional and new AI-specific patterns

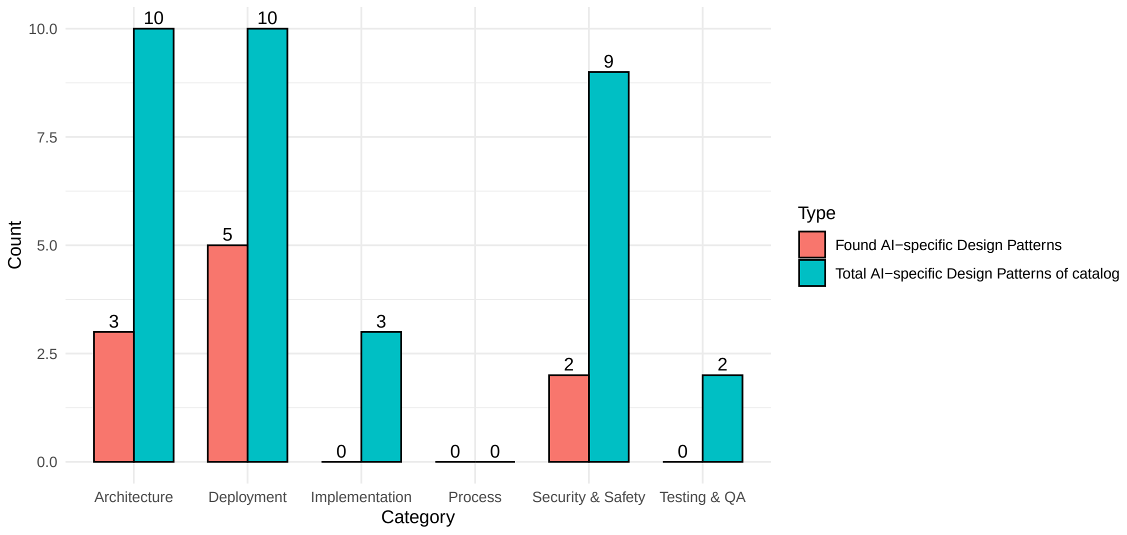

Figure 5.6: Comparison of the number of found AI-specific patterns per category

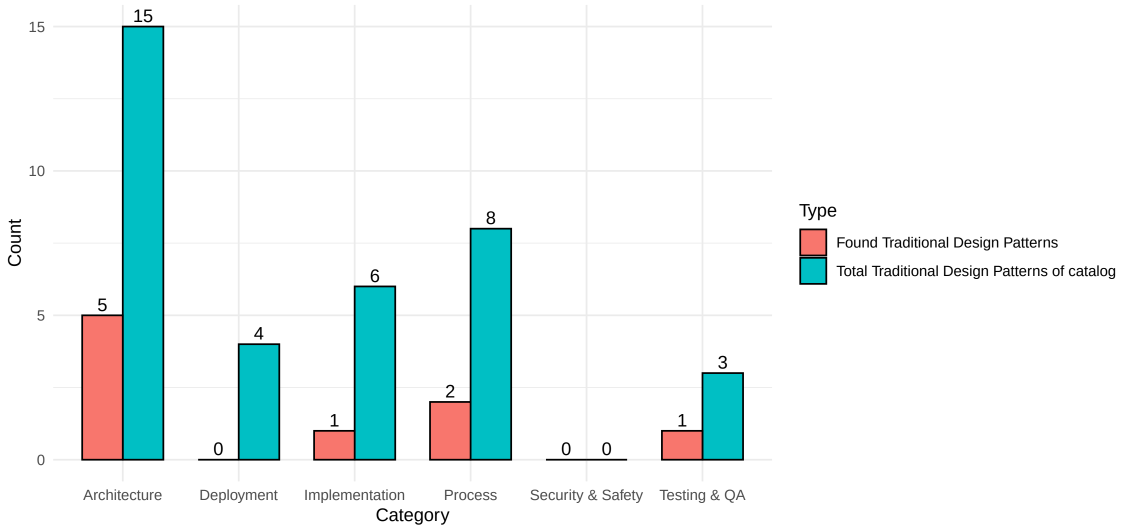

Figure 5.7: Comparison of the number of found adapted traditional patterns per category

6 Discussion

This chapter discusses the collected results of this work, including the repository collection and the identified design patterns and architectural tactics, and also addresses our research questions. We try to draw conclusions about what design patterns and architectural characteristics we found in the projects we analyzed.

6.1 Repository Collection

After querying the GitHub Search API with our given parameters and filtering out repositories based on other parameters that we could not already filter through the GitHub Search API query, we get 37 repositories which we classify as AI-based systems after manual selection, which is only 11.9% of our total repositories before our manual selection.

Similar to our study, Gonzalez et al. [GZN20] also collected AI & ML repositories. They used the GitHub Search API to access such repositories as we do, with the difference, that we used only four topics (*ai*, *artificial-intelligence*, *ml*, *machine-learning*) and Gonzalez et al. queried the GitHub API for additional topics related to *artificial intelligence*, *deep learning* and *machine learning*, which provided them with 439 topic labels. Only then they queried the GitHub Search API for repositories containing at least one of these 439 topics.

Like us, they categorized each resulting AI & ML repository as *Applied* or *Tool*, where *Applied* in our case would reflect the *AI-based System* category, and *Tool* would reflect the categories *Library*, *Framework* and *Tool*. Also like us, the authors also set exclusion criteria for the repositories, similar to us, such as a minimum of 5 stars and 5 forks, the last commit could not be too far back, and it had to be a software project.

Comparing the results, Gonzalez et al. identified 4524 *Applied* AI & ML and 700 *Tool* AI & ML repositories, which makes 86% Applied and 14% Tool Repositories. If we add up the repositories of our three categories *Library*, *Framework* and *Tool*, we get 228 AI & ML *Tool* repositories and 37 *Aplied* repositories, which is 86% *Tool* and 14% *Aplied* repositories, which is quite the opposite of the distribution of Gonzalez et al.

However, we used different parameters to search the GitHub Search API. Gonzalez et al. had a broader selection of topic tags, while we only queried with our four topic tags. We also requested 20 forks instead of 5 and another big difference is that we filtered out all repositories that did not have at least 30 contributors. Gonzalez et al. [GZN20] compared the number of committers in *Applied* and *Tool* repositories and found a difference in the number of committers in *Applied* and *Tool* repositories, as shown in Figure 6.1, with a mean of 1 committer in *Applied* repositories and a

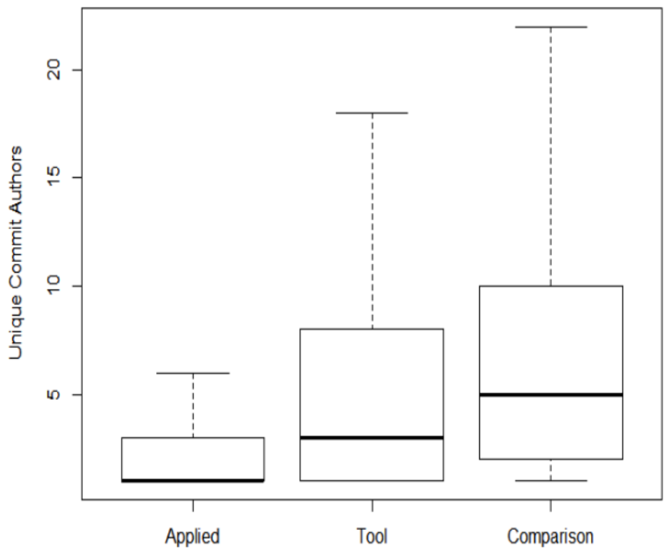

Figure 6.1: Unique Commit Authors of Applied, Tool and a set of comparison repositories (outliers omitted) [GZN20]

mean of 5 committers in *Tool* repositories. Of course, outliers with particularly large numbers of committers are not shown in this figure. This difference may explain why the distribution of our *Applied* and *Tool* repositories is so large compared to Gonzalez et al. [GZN20].

For all of our AI-based systems, we have analyzed their documentation according to architectural documentation. Of our 37 AI-based system repositories, 29 repositories had basically no architectural documentation in their project documentation, which is 78% of AI-based repositories. Ding et al. [DLT+14] analyzed OSS projects from OSS sources such as GitHub, but also Sourceforge, Google Code and Tigris, with the finding, that only 5.4% of the projects have documentation about their software architecture. Out of our 8 repositories where we found architectural documentation in their project documentation, we ranked only 2 projects with good architectural documentation and the other 6 with just basic architectural documentation, as shown in Table 5.2.

6.2 RQ1: How are design patterns and architectural tactics applied in AI-based software systems?

With the identification of 19 design patterns from the catalog of Heiland et al. [HHB23] in our three AI-based software systems we analyzed, we can draw conclusions according to the distribution of the categories of the design patterns found. The design patterns of the *Architecture* and *Deployment* categories already make up over half of the design pattern catalog with 39 out of 70 patterns, and they also outweigh the total number of design patterns found with 13 design patterns from these categories of the total 19 design patterns found in the three projects, which accounts for over 68%. In contrast, the remaining design patterns found in the *Implementation*, *Process*, *Security & Safety* and *Testing & Quality Assurance* categories are very low in relative comparison to the total design patterns in the catalog.

Two of our three systems were web applications, both of which followed the typical *Client-Server* design pattern. You can also see that both web applications had the actual logic of the application and the ML models separated in their backends, in their own small services. Open-Assistant has extended this with the *Parameter-Server Abstraction* design pattern by distributing and outsourcing the workload to worker nodes. This shows that the AI components are not only moved to the backend, but are also often separated from the business logic itself into their own services.

The large number of *Deployment* design patterns is due to the fact that 4 out of 5 of the design patterns found are 4 different ways of deploying the models. This shows that already many methods of model deployment are used in our few three projects.

We did not find many design patterns from the *Implementation* category, which could be due to the fact that these are harder to find, as they are more hidden in the source code than other design patterns and therefore harder to identify.

In the area of *Security & Safety*, we have only seen something in Open-Assistant, in which a separate service for safety matters has been deployed, which promptly checks user entries to see whether they violate its own safety guidelines. This seems to us to be an indication that *Security & Safety* are being neglected in the development of AI-based software systems. Serban and Visser [SV21] made similar findings, conducting interviews with practitioners and surveys with software architects and found that architectural decision drivers for trustworthy ML, such as *Security*, were not considered important by respondents. However, *Security & Safety* are more of a system property, which is why this is probably more difficult to capture in the form of a design pattern. Developers may try to solve *Security & Safety* at system level and not at design pattern level.

We may have missed design patterns due to our study design as discussed in Section 6.4. This may lead to underreporting for some categories. This seems especially likely for the *Process* category, since many of the patterns would require insight into the development process itself, and could only be identified if we were part of the development process (which we were not), or if it was documented somewhere. For example, the *Internal Feedback* design pattern can only be identified if the documentation or public project management tools indicate that the product was tested by internal users.

We see many similar findings to the study by Nahar et al. [NZL+23] and our findings for the projects we analyzed. Below is a list of some of the findings from Nahar et al. that we also find in our three projects.

Many products rely on third-party ML models (#6): In the three projects we analyzed, we saw several third-party ML models, such as including models via libraries (such as face_recognition), external APIs (such as OpenAI API), or by loading pre-trained model files, mostly from GitHub releases. But we have also seen self-trained models, for example in the Open-Assistant project, but also in a small service in LibrePhotos.

Most products use raw model predictions without any post-processing (#9): In the projects we analyzed, we did not see any post-processing of model predictions that further validated the predictions.

It is common to use multiple models in a product, though mostly the models work indepedently of each other (#11): In Librephotos, different backends were created for different ML tasks. Different models are used, which fulfill their tasks independently of each other.

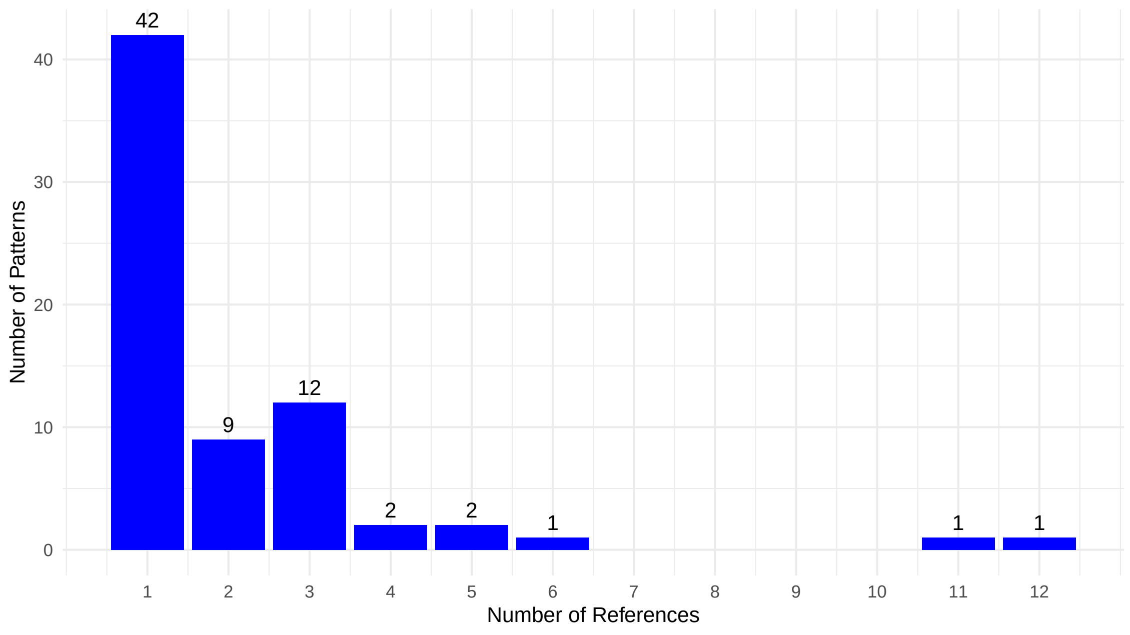

Figure 6.2: Number of patterns per received number of references by resources identified by Heiland et al. [HHB23]

Pipeline automation is not common in open-source ML products (#12): Open-Assistant and a small service in LibrePhotos were the only services that trained their models themselves, neither of which was automated.

Testing regular software functionality is common. Model testing is notable scarce. Data validation is rare. (#17): We also found that while the common, non-AI parts of the project were thoroughly tested, for example with unit tests, the AI components and models were not tested at all.

We also compared the number of references per design pattern from [HHB23] (Figure 6.2) with the number of references from the design patterns we found (Figure 6.3). One might assume that the more references a design pattern has, the more often it might be used in projects. However, if we compare Figure 6.2 and Figure 6.3, we can see that the distribution looks relatively the same and therefore no significant difference can be found.

6.3 RQ2: How do design patterns in AI-based software systems differ from those in traditional software systems?

We have seen that the design patterns found are relatively balanced between traditionally adapted design patterns (9) and new AI-specific design patterns (10). This also shows that traditional design patterns, such as the *Client-Server* design pattern, are nothing new and continue to be used here, while also, for example, the *Infrastructure as Code* design pattern is used in AI-based systems, which need to be extended from traditional systems in terms of how to build and run the model in production.

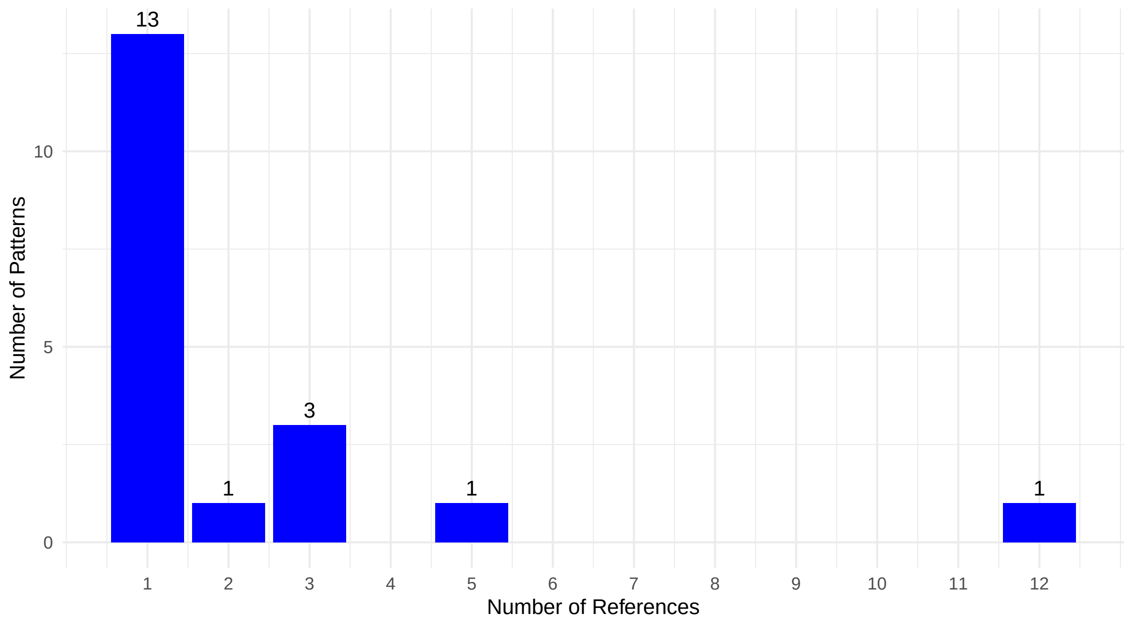

Figure 6.3: Number of patterns per received number of references by resources of found design patterns

If we look at the distribution of design patterns by category per traditional adapted and new AI-specific design patterns, we can see that a large proportion of *Architecture* patterns are adapted from traditional systems, suggesting that the rough architecture of traditional and AI-based systems is similar.

One major point of difference between AI-based systems and traditional systems is how ML models are deployed and used.

All design patterns in the *Process* category are traditional adapted design patterns. Despite the already mentioned limitation that many design patterns of the category are difficult to detect in a system because they are used during the development phase itself and some may not be recognizable from the source code or are not mentioned in the documentation, we found two design patterns in the category, so we see that traditional adapted design patterns are also used in AI-based systems.

On the other hand, *Security & Safety* is a category consisting only of new AI-specific design patterns, for which we found only 2 design patterns in one project.

We only saw test cases in one project where the model is tested to see if it returns an expected result, and these tests were kept very simple. We could not see more than this only traditional adapted design pattern in the *Testing & Quality Assurance* category for the analyzed projects. It is already known that ML systems are difficult to test because the answers to specific questions do not yet exist to test the correctness of a ML model [MKA07]. Much of the literature on testing ML systems attempts to find techniques to solve this problem, often relying on traditional software techniques [ZHML22]. In interviews with practitioners, Nahar et al. [NZLK22] found that most organizations do not perform monitoring or online testing because they find it difficult.

6.4 Threats to Validity

To curate a diverse and representative collection of AI-based software systems, we have chosen to use GitHub, a platform that hosts over 420 million git repositories. This may mean that we cover just a limited number of repositories, as GitHub is not the only platform hosting code repositories. But GitHub's prominence and extensive repository coverage made it our primary source, given its popularity for hosting public projects. By using GitHub, we ensured access to a wide range of AI-based systems, taking advantage of its large user base and variety of available projects.

Identifying appropriate repositories has been accomplished through the use of topic tags that developers assign to their projects. However, this strategy introduces potential biases, in particular excluding relevant projects that do not use the specific tags we are searching for. Nevertheless, this approach allowed us to compile a broad list of AI-based software systems.

To further filter our selection, we established criteria that focused on community engagement and project activity. A minimum of 20 forks was required to filter out less significant projects, with the assumption that GitHub's "fork and pull" model represents community interest. We also required a minimum of 30 contributors and 300 commits to ensure that projects were both properly sized and active. We are aware that we could miss out on relevant projects, but we specifically wanted projects that have a certain size and community interest. These criteria served to increase the quality of our repository collection by targeting software systems with a demonstrated level of community interest and ongoing development.

The categorization and final selection of projects required relying on the project descriptions, READMEs, and documentation. This reliance was crucial for accurately determining whether the project used AI and in what way, and for categorizing a large number of repositories without spending too much effort, since the number of repositories for manual selection was quite high. Sorting repositories based on architectural documentation and popularity allowed us to prioritize systems with significant community value and well-documented architectures, ensuring the relevance and quality of our repository collection.

Since the thesis is an individual performance, the analysis of the projects was carried out by one person and could contain bias. Other people analyzing the projects may identify other design patterns that we may have missed because of this bias. However, the results were discussed with my supervisor, but this may not eliminate the bias.

The identification of design patterns in the selected projects was performed using a design pattern catalog for AI-based systems by Heiland et al. [HHB23]. While we carefully analyzed the projects, we may have missed some design patterns that were used in the projects, but we were unable to identify them. In order to identify some design patterns, they would also have to be mentioned in documentation or public project management tools to determine whether a particular design pattern was used. However, if we had conducted interviews with the developers of the projects, we might have gained more insight into certain categories.

For reasons of time and scope, we were only able to analyze three projects, which may limit the generalization of the results. Although there are tools for automatically identifying design patterns, we intentionally chose not to use them, first because they can be error-prone, and second because most of the new AI-specific design patterns may not be identified by the tools. However, given the timeframe of the project, this risk is unavoidable.

7 Conclusion

This thesis presented the results of analyzing the software architecture of three open-source AI-based software systems to provide insights into what the software architecture of AI-based systems looks like, with a particular focus on the design patterns and architectural tactics used.

Initially, AI-based systems from GitHub were mined for this analysis. In particular, the analysis used a catalog of design patterns for AI-based software systems to identify the design patterns within the analyzed systems. From the three AI-based systems analyzed, we were able to identify a total of 19 unique design patterns out of 70 total design patterns provided by the design pattern catalog. A significant number of the design patterns found belonged to the *Architecture* and *Deployment* categories. In distinguishing between traditional adapted design patterns and new AI-specific design patterns, it was observed that many found *Architecture* design patterns are traditional adapted design patterns, while many *Deployment* design patterns are new AI-specific, which is because most deployment patterns care about how the ML models are deployed.

The use of AI in these systems has led to the emergence of many new AI-specific design patterns for *Security & Safety*, although only a few of these design patterns were identified in our analysis. This seems to indicate a lack of awareness of these issues among practitioners. Therefore, as there is already research in this area, but it seems to have limited applicability [SV21], we recommend further research attention for bringing *Security & Safety* techniques into practice.

Another finding is that while the traditional parts of the systems were well tested, the AI components were very poorly tested. While software testing for AI-based software systems is quite prevalent in research [MBF+22], the problem seems to be how to put the research into practice. For this reason, we also recommend further research on how to put the knowledge gained from research into practice for *Testing & Quality Assurance*.

Finally, we also recommend conducting a study like this again, with a larger team and more repositories to reduce potential bias. With a larger number of repositories, we can more clearly and generally interpret the results for AI-based software systems. In addition, a larger team can cross-check the results, which unfortunately was not possible here on a large scale, which also minimizes bias.

Bibliography

[ABBS15] M. Allamanis, E. T. Barr, C. Bird, C. Sutton. “Suggesting accurate method and class names”. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ESEC/FSE 2015. Bergamo, Italy: Association for Computing Machinery, 2015, pp. 38–49. ISBN: 9781450336758. DOI: 10.1145/2786805.2786849 (cit. on p. 24).

[APS16] M. Allamanis, H. Peng, C. Sutton. “A Convolutional Attention Network for Extreme Summarization of Source Code”. In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by M. F. Balcan, K. Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, June 2016, pp. 2091–2100. URL: https://proceedings.mlr.press/v48/allamanis16.html (cit. on p. 24).

[AW07] S. A. Ajila, D. Wu. “Empirical study of the effects of open source adoption on software development economics”. In: *Journal of Systems and Software* 80.9 (2007). Evaluation and Assessment in Software Engineering, pp. 1517–1529. ISSN: 0164-1212. DOI: 10.1016/j.jss.2007.01.011 (cit. on p. 17).

[AZ11] F. Arcelli Fontana, M. Zanoni. “A tool for design pattern detection and software architecture reconstruction”. In: *Information Sciences* 181.7 (2011), pp. 1306–1324. ISSN: 0020-0255. DOI: 10.1016/j.ins.2010.12.002 (cit. on p. 26).

[BME19] L. Bernardi, T. Mavridis, P. Estevez. “150 Successful Machine Learning Models: 6 Lessons Learned at Booking.com”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD ’19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 1743–1751. ISBN: 9781450362016. DOI: 10.1145/3292500.3330744 (cit. on p. 15).

[BRG17] B. Bafandeh Mayvan, A. Rasoolzadegan, Z. Ghavidel Yazdi. “The state of the art on design patterns: A systematic mapping of the literature”. In: *Journal of Systems and Software* 125 (2017), pp. 93–118. ISSN: 0164-1212. DOI: 10.1016/j.jss.2016.11.030 (cit. on p. 19).

[CGL+03] P. Clements, D. Garlan, R. Little, R. Nord, J. Stafford. “Documenting software architectures: views and beyond”. In: *25th International Conference on Software Engineering, 2003. Proceedings.* 2003, pp. 740–741. DOI: 10.1109/ICSE.2003.1201264 (cit. on p. 17).

[DLT+14] W. Ding, P. Liang, A. Tang, H. Van Vliet, M. Shahin. “How Do Open Source Communities Document Software Architecture: An Exploratory Survey”. In: *2014 19th International Conference on Engineering of Complex Computer Systems*. 2014, pp. 136–145. DOI: 10.1109/ICECCS.2014.26 (cit. on pp. 17, 36).

[GA08] Y.-G. Guéhéneuc, G. Antoniol. “DeMIMA: A Multilayered Approach for Design Pattern Identification”. In: *IEEE Transactions on Software Engineering* 34.5 (2008), pp. 667–684. DOI: 10.1109/TSE.2008.48 (cit. on p. 26).

[Gar00] D. Garlan. “Software architecture: a roadmap”. In: *Proceedings of the Conference on The Future of Software Engineering*. ICSE ’00. Limerick, Ireland: Association for Computing Machinery, 2000, pp. 91–101. ISBN: 1581132530. DOI: 10.1145/336512.336537 (cit. on p. 17).

[GFM16] V. Garousi, M. Felderer, M. V. Mäntylä. “The Need for Multivocal Literature Reviews in Software Engineering: Complementing Systematic Literature Reviews with Grey Literature”. In: *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*. EASE ’16. Limerick, Ireland: Association for Computing Machinery, 2016. ISBN: 9781450336918. DOI: 10.1145/2915970.2916008 (cit. on p. 19).

[GHJV95] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design patterns: elements of reusable object-oriented software*. USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0201633612 (cit. on pp. 17, 26).

[GS12] G. Gousios, D. Spinellis. “GHTorrent: Github’s data from a firehose”. In: *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*. 2012, pp. 12–21. DOI: 10.1109/MSR.2012.6224294 (cit. on p. 22).

[GTS+22] K. Grotov, S. Titov, V. Sotnikov, Y. Golubev, T. Bryksin. “A large-scale comparison of Python code in Jupyter notebooks and scripts”. In: *Proceedings of the 19th International Conference on Mining Software Repositories*. MSR ’22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 353–364. ISBN: 9781450393034. DOI: 10.1145/3524842.3528447 (cit. on p. 22).

[GZN20] D. Gonzalez, T. Zimmermann, N. Nagappan. “The State of the ML-Universe: 10 Years of Artificial Intelligence & Machine Learning Software Development on GitHub”. In: *Proceedings of the 17th International Conference on Mining Software Repositories*. MSR ’20. Seoul, Republic of Korea: Association for Computing Machinery, 2020, pp. 431–442. ISBN: 9781450375177. DOI: 10.1145/3379597.3387473 (cit. on pp. 20, 22–24, 35, 36).

[HBB+18] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, X. Wang. “Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective”. In: *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2018, pp. 620–629. DOI: 10.1109/HPCA.2018.00059 (cit. on p. 15).

[HCHV21] M. Haakman, L. Cruz, H. Huijgens, A. Van Deursen. “AI lifecycle models need to be revised: An exploratory study in Fintech”. In: *Empirical Software Engineering* 26.5 (2021), p. 95. DOI: 10.1007/s10664-021-09993-1 (cit. on p. 15).

[HHB23] L. Heiland, M. Hauser, J. Bogner. *Design Patterns for AI-based Systems: A Multivocal Literature Review and Pattern Repository*. 2023. DOI: 10.48550/arXiv.2303.13173. arXiv: 2303.13173 [cs.SE] (cit. on pp. 17, 19, 24, 26, 29, 31, 36, 38, 40).

[IY19] F. Ishikawa, N. Yoshioka. “How Do Engineers Perceive Difficulties in Engineering of Machine-Learning Systems? - Questionnaire Survey”. In: *2019 IEEE/ACM Joint 7th International Workshop on Conducting Empirical Studies in Industry (CESI) and 6th International Workshop on Software Engineering Research and Industrial Practice (SER&IP)*. 2019, pp. 2–9. DOI: 10.1109/CESSER-IP.2019.00009 (cit. on p. 15).

[JLH+17] J. Jiang, D. Lo, J. He, X. Xia, P. S. Kochhar, L. Zhang. “Why and how developers fork what from whom in GitHub”. In: *Empirical Software Engineering* 22 (2017), pp. 547–578. DOI: 10.1007/s10664-016-9436-6 (cit. on p. 23).

[KGB+14] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, D. Damian. “The promises and perils of mining GitHub”. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: Association for Computing Machinery, 2014, pp. 92–101. ISBN: 9781450328630. DOI: 10.1145/2597073.2597074 (cit. on pp. 21, 24).

[KS20] R.-M. Karampatsis, C. Sutton. “How Often Do Single-Statement Bugs Occur? The ManySStuBs4J Dataset”. In: *Proceedings of the 17th International Conference on Mining Software Repositories*. MSR ’20. Seoul, Republic of Korea: Association for Computing Machinery, 2020, pp. 573–577. ISBN: 9781450375177. DOI: 10.1145/3379597.3387491 (cit. on p. 24).

[LOX21] G. A. Lewis, I. Ozkaya, X. Xu. “Software Architecture Challenges for ML Systems”. In: *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2021, pp. 634–638. DOI: 10.1109/ICSME52107.2021.00071 (cit. on p. 15).

[Mar00] R. C. Martin. “Design principles and design patterns”. In: *Object Mentor* 1.34 (2000), p. 597. URL: https://labs.cs.upt.ro/labs/ip2/html/lectures/2/res/Martin-PrinciplesAndPatterns.PDF (cit. on p. 17).

[MBF+22] S. Martínez-Fernández, J. Bogner, X. Franch, M. Oriol, J. Siebert, A. Trendowicz, A. M. Vollmer, S. Wagner. “Software Engineering for AI-Based Systems: A Survey”. In: *ACM Trans. Softw. Eng. Methodol.* 31.2 (Apr. 2022). ISSN: 1049-331X. DOI: 10.1145/3487043 (cit. on pp. 15, 18, 19, 41).

[MKA07] C. Murphy, G. E. Kaiser, M. Arias. “An approach to software testing of machine learning applications”. In: (2007). DOI: 10.7916/D8R49ZNR (cit. on p. 39).

[NMY+20] T. Nakamaru, T. Matsunaga, T. Yamazaki, S. Akiyama, S. Chiba. “An Empirical Study of Method Chaining in Java”. In: *Proceedings of the 17th International Conference on Mining Software Repositories*. MSR ’20. Seoul, Republic of Korea: Association for Computing Machinery, 2020, pp. 93–102. ISBN: 9781450375177. DOI: 10.1145/3379597.3387441 (cit. on p. 22).

[NZL+23] N. Nahar, H. Zhang, G. Lewis, S. Zhou, C. Kästner. “A dataset and analysis of open-source machine learning products”. In: *arXiv preprint arXiv:2308.04328* (2023). DOI: 10.48550/arXiv.2308.04328 (cit. on pp. 19, 22, 37).

[NZLK22] N. Nahar, S. Zhou, G. Lewis, C. Kästner. “Collaboration challenges in building ML-enabled systems: communication, documentation, engineering, and process”. In: *Proceedings of the 44th International Conference on Software Engineering*. ICSE ’22. Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 413–425. ISBN: 9781450392211. DOI: 10.1145/3510003.3510209 (cit. on pp. 15, 39).

[RRK+19] M. S. Rahman, E. Rivera, F. Khomh, Y.-G. Guéhéneuc, B. Lehnert. "Machine learning software engineering in practice: An industrial case study". In: *arXiv preprint arXiv:1906.07154* (2019). doi: 10.48550/arXiv.1906.07154 (cit. on p. 15).

[SV21] A. Serban, J. Visser. "An empirical study of software architecture for machine learning". In: *arXiv preprint arXiv:2105.12422* 39 (2021) (cit. on pp. 15, 20, 37, 41).

[TLP18] D. Taibi, V. Lenarduzzi, C. Pahl. "Architectural patterns for microservices: a systematic mapping study". In: *CLOSER 2018: Proceedings of the 8th International Conference on Cloud Computing and Services Science; Funchal, Madeira, Portugal, 19-21 March 2018*. SciTePress. 2018 (cit. on p. 17).

[VB19] A. Vogelsang, M. Borg. "Requirements Engineering for Machine Learning: Perspectives from Data Scientists". In: *2019 IEEE 27th International Requirements Engineering Conference Workshops (REW)*. 2019, pp. 245–251. doi: 10.1109/REW.2019.00050 (cit. on p. 15).

[WKG+22] H. Washizaki, F. Khomh, Y.-G. Guéhéneuc, H. Takeuchi, N. Natori, T. Doi, S. Okuda. "Software-Engineering Design Patterns for Machine Learning Applications". In: *Computer* 55.3 (2022), pp. 30–39. doi: 10.1109/MC.2021.3137227 (cit. on p. 19).

[WOH+20] H. Washizaki, S. Ogata, A. Hazeyama, T. Okubo, E. B. Fernandez, N. Yoshioka. "Landscape of Architecture and Design Patterns for IoT Systems". In: *IEEE Internet of Things Journal* 7.10 (2020), pp. 10091–10101. doi: 10.1109/JIOT.2020.3003528 (cit. on p. 17).

[WUKG19] H. Washizaki, H. Uchida, F. Khomh, Y.-G. Guéhéneuc. "Studying Software Engineering Patterns for Designing Machine Learning Systems". In: *2019 10th International Workshop on Empirical Software Engineering in Practice (IWESEP)*. 2019, pp. 49–495. doi: 10.1109/IWESEP49350.2019.00017 (cit. on p. 19).

[ZHML22] J. M. Zhang, M. Harman, L. Ma, Y. Liu. "Machine Learning Testing: Survey, Landscapes and Horizons". In: *IEEE Transactions on Software Engineering* 48.1 (2022), pp. 1–36. doi: 10.1109/TSE.2019.2962027 (cit. on p. 39).

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted hard copies.

place, date, signature