University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Evaluation of a Large Language Model-powered Threat Modeling Approach with Applications to AI Security

Raphael Wäspi
St. Gallen, Switzerland
Student ID: 18-918-938

Supervisor: Jan von der Assen, Chao Feng
Date of Submission: March 10, 2025

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Threat modeling is an important technique in the field of cybersecurity that helps to systematically identify and mitigate risks in systems. As systems based on artificial intelligence (AI) become increasingly complex, existing approaches to threat modeling often require extensive documentation, which makes threat modeling impractical in early stages of development. This thesis investigates the feasibility of using large language models to support threat modeling for AI systems, with a focus on threat identification and validation with minimal inputs. A comprehensive literature review revealed that existing solutions are primarily based on detailed system documentation and do not offer an approach that works only with minimal inputs such as data-flow diagrams.

To address this gap, this thesis presents a prototype that uses Retrieval Augmented Generation to identify and validate AI threats using only data-flow diagrams as input. The system is designed with a two-stage approach that separates the threat identification and validation processes. Furthermore, this thesis investigates different Retrieval Augmented Generation configurations and evaluates the performance of different large language models in these processes. This evaluation is conducted through a number of interviews that provide information on which combinations of Retrieval Augmented Generation configurations and large language models provide the most effective results. This thesis demonstrates that threat modeling based on an large language model approach can be a valuable for identifying and validating security threats in AI systems. In addition, it provides a practical guidance on optimal configurations for such a tool and gives examples for future research in this field.

Zusammenfassung

Die Bedrohungsmodellierung ist eine wichtige Technik im Bereich der Cybersicherheit, die dabei hilft, Risiken in Systemen systematisch zu erkennen und zu reduzieren. Da Systeme, die auf künstlicher Intelligenz basieren, immer komplexer werden, erfordern bestehende Ansätze zur Bedrohungsmodellierung oft eine umfangreiche Dokumentation. Dies führt dazu, dass eine Modellierung in frühen Entwicklungsphasen nicht geeignet ist. In dieser Arbeit wird die Umsetzbarkeit der Nutzung von Large Language Models zur Unterstützung der Bedrohungsmodellierung für Systeme der künstlichen Intelligenz untersucht, wobei der Schwerpunkt auf der Erkennung und Validierung von Bedrohungen mit minimalen Inputs liegt. Eine umfassende Literaturrecherche ergab, dass bestehende Lösungen in erster Linie auf einer detaillierten Systemdokumentation beruhen und keinen Ansatz bieten, der nur mit minimalen Inputs wie Datenflussdiagrammen funktioniert.

Um diese Lücke zu schliessen, wird in dieser Arbeit ein Prototyp präsentiert, der Retrieval Augmented Generation einsetzt. Damit sollen Bedrohungen erkannt und validiert werden, wobei lediglich ein Datenflussdiagramm als Input dient. Das System ist mit einem zweistufigen Ansatz konzipiert, der die Erkennung und Validierung von Bedrohungen trennt. Darüber hinaus werden in dieser Arbeit verschiedene Retrieval Augmented Generation Konfigurationen untersucht und die Leistung verschiedener Sprachmodellen in diesen Prozessen evaluiert. Diese Evaluation wird mittels Interviews durchgeführt und es wird geprüft, welche Kombinationen von Retrieval Augmented Generation Konfigurationen und Sprachmodellen die besten Ergebnisse liefern. Diese Arbeit zeigt, dass eine auf grossen Sprachmodellen basierende Bedrohungsmodellierung ein wertvolles Instrument zur Identifizierung und Validierung von Sicherheitsbedrohungen in Systemen mit künstlicher Intelligenz sein kann. Darüber hinaus bietet die Thesis einen praktischen Anhaltspunkt für die optimale Konfiguration einer solchen Applikation und gibt Beispiele für die künftige Forschung in diesem Bereich.

Acknowledgments

I would like to express my sincere gratitude to my supervisors and everyone who took the time to participate in the interviews or supported me in any way. In particular, I would like to thank Jan von der Assen for his continuous support and valuable discussions throughout this Master Thesis. His broad experience and expert feedback were of great value to this thesis.

I would also like to thank Prof. Dr. Burkhard Stiller for the possibility to complete my Master Thesis at the Communication Systems Group (CSG) of the University Zurich.

Contents

Chapter 1

Introduction

In the context of cybersecurity, threat modeling is an important approach that allows threats to be identified before or during the development process. It is also a structured approach with the aim of developing techniques to mitigate these identified threats. In recent years, several techniques have been created to improve the identification and mitigation of threats. In addition, threat modeling is often used for secure software development, risk assessment, or to promote security awareness [1].

Another change in software architecture is the rise of Artificial Intelligence (AI) with its diverse methods (*e.g.*, Machine Learning, Deep Learning), as more and more data is available and decisions are now determined by data instead of simple logic. For this reason, research has identified new threats that are relevant for such applications [2]. New methods and knowledge bases (KBs) are therefore required to identify these threats using threat models.

However, creating threat models for AI systems is a difficult task for software engineers and data scientists [3]. This is due to several challenges. First, research has focused excessively on the various threats instead of developing threat modeling. Second, existing threat modeling frameworks have been developed for traditional software and not for AI software, which may have AI-specific threats.

However, creating threat models for AI systems is a difficult task for software engineers and data scientists [3]. This is due to a number of challenges, which for example is the fact that research has focused on the various threats rather than developing threat modeling. Furthermore, the existing threat modeling frameworks have been developed for traditional software and not for AI software, which may have AI-specific threats.

This thesis investigates the feasibility of using an Large Language Model (LLM) approach to support AI threat modeling. In particular, it investigates whether a prototype is able to identify and validate AI security threats with minimal inputs such as Data-flow Diagrams (DFDs). This thesis contributes to research by evaluating the feasibility of automating AI threat models with an LLM approach and provides insights into a practical LLM application in cybersecurity.

1.1 Motivation

AI systems contain different security threats than traditional software, which cannot be detected using traditional threat models. This becomes even more challenging when trying to detect these AI threats in early stages of development where only limited data is available that can be used for identification, such as DFDs. Currently, there is little research that addresses how LLMs can assist in AI threat modeling. The research that does exist in this area tends to rely on extensive documentation or intermediate structures, which adds complexity at early stages of development. This research gap leaves a key question unanswered: can an LLM-based approach be used to detect and validate security threats in AI systems with only minimal inputs such as DFDs?

1.2 Description of Work

This thesis proposes an LLM-based approach that identifies and validates threats directly from DFDs without relying on intermediary structures. To address the research gap, this work contributes by developing a prototype that uses an LLM to detect and validate threats and demonstrates a practical application of LLMs in AI security. Furthermore, it evaluates whether minimal inputs are sufficient for meaningful threat identification, considering the limitations of an early development phase where detailed documentation is often not available.

By separating threat identification and validation into two distinct steps, this study also examines their individual effectiveness. By separating these processes, the individual effectiveness of each step will be evaluated, as a combined process of poor quality would not allow a clear understanding of whether threat identification or threat validation is working as intended. Considering that KBs in the real world are large and often exceed the capacity of an LLM input window, this thesis also investigates the integration of Retrieval Augmented Generation (RAG). If successful, this work could push the automation of early-stage AI threat modeling and make threat modeling more accessible.

1.3 Thesis Outline

The first chapter introduces this thesis and gives an overview of the motivation, the goals, and the work carried out. It answers the key questions of why this research is important and what was done to achieve its objectives. Chapter 2 provides the basic concepts required to understand this thesis and covers the topics of threat modeling, AI security threats, LLMs, and RAG. Chapter 3 provides an overview of existing research on LLMs in cybersecurity, AI for threat modeling, and LLM-based threat modeling. Based on this analysis, the current limitations and opportunities are identified. Chapter 4 presents the architecture of the prototype, which explains how its components interact and how the principles of RAG are integrated into the system. It also shows the used technologies for the prototype implementation. This highlights key aspects of the source code, critical

design decisions, and the rationale behind some implementation choices. Chapter 5 describes the evaluation process, where the two main approaches (threat identification and threat validation) are evaluated separately based on qualitative metrics and performance. In addition, different prototype configurations are compared to determine which configuration performs best with minimal inputs. Finally, the last chapter summarizes the key findings, discusses their relevance, and identifies possible directions for future work that could build on this work.

Chapter 2

Background

To understand the theoretical concepts presented in this master's thesis, this chapter introduces the basic concepts and terminology. The first section explains what threat modeling is and how it is used in modern software development. This is followed by an introduction to AI security threats and LLMs. Finally, the chapter contains a short overview of the RAG concept and its key principles.

2.1 Threat Modeling

Threats can be described as potential damage that may occur in the future [4]. The reason for their occurrence is usually a weakness in the design of the application. To avoid such vulnerabilities, threat modeling can be used to provide appropriate mitigation techniques in the design. Such a mitigation at the beginning of development can prevent a much more costly fix at a later stage [5], an approach known as DevSecOps. But even if a mitigation is not possible, additional security controls or measures can be implemented to reduce the risk of the threat being exploited [6].

The Threat Modeling Manifesto [7] outlines four key principles for effective threat modeling:

- Threat modeling is most effective when it is used to improve the security and privacy of a system through frequent analysis at an early stage of the development process.
- Threat modeling should be integrated into the company's development cycle to ensure that it complements existing workflows and processes.
- The outcome of the threat model is important if it is of value to the stakeholder.
- Dialogues help to build a common understanding that leads to values, while documents help to document this understanding and enable measurement.

By integrating these principles, threat modeling systematically addresses threats, mitigates vulnerabilities early and reduces costly rework later in development.

The Open Worldwide Application Security Project (OWASP) [8] structures the process in four steps. First, the scope of the system must be assessed. This can be an entire application or just a major change in a sprint. In the second threat modeling step, it is necessary to determine what can go wrong in the system. This can be done through a structured process such as STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege), Attack Trees or through simple brainstorming. Once the threats are identified, the third step is to decide what remedial action can be taken to eliminate or mitigate the threat. In a final step, the work should be evaluated and questioned whether the work has been done well enough.

However, in the software development environment of today, where applications are constantly evolving, it is not enough to perform threat modeling once [9]. Continuous development can lead to additional threats that were not present at the beginning of the software development cycle. Therefore, continuous threat modeling is created that captures the system in the initial phase and then continues as the software evolves. This reveals threats as they emerge, evolve and change. The OWASP [8] recommends creating a threat model after events such as a new feature release, a security incident, or a change in infrastructure or architecture.

2.2 Security Threats in Artificial Intelligence

With the quick growth of modern AI systems, concerns about security and data protection are growing [10]. This has led to the creation of AI security, which is defined as tools, strategies, and processes for detecting and preventing threats and vulnerabilities. Such attacks can compromise the confidentiality, integrity or availability of an AI model or AI-systems. Therefore, identifying and mitigating vulnerabilities in AI systems is a critical component of AI security. AI developers must ensure their system does not contain vulnerabilities and implement controls to protect and respond to fraudulent behavior that targets the system environment or its users. [11]. To prevent new forms of exploitation, it is important to investigate new defense mechanisms and threat modeling approaches.

However, there are already several efforts and initiatives that focus on AI-related attacks [3]. MITRE Adversarial Threat Landscape for AI Systems (ATLAS), for example, provides an overview of attacks that are tailored to AI systems and offers a structured framework for understanding and mitigating AI-specific threats [10]. An example of such adversarial attacks and their potential impact on AI-systems can be seen in Table 2.1. Similarly, Microsoft has proposed a KB that provides guidelines for identifying and mitigating vulnerabilities in AI systems [11]. The OWASP has also published resources for ensuring the security of AI-dependent systems [12]. In addition, the European Union Agency for Cybersecurity (ENISA) has published a comprehensive report detailing AI-related threats [13]. Together, these efforts help to gain an understanding of AI security concerns and develop robust countermeasures.

Attack	Overview
Poisoning Attack	An attacker changes the training data of an AI system in order to receive a desired result. By manipulating the training data, an attacker can implement backdoors in the model through which an input with the trigger leads to a certain output.
Evasion Attack	Attacker elicits an incorrect response from a model by crafting adversarial inputs. Typically, these inputs are designed to be indistinguishable from normal data. These attacks can be targeted, where the attacker tries to produce a specific classification, or untargeted, where they attempt to produce any incorrect classification.
Functional Extraction	The attacker gets an equivalent model by querying the model iteratively. This enables an attacker to analyze the offline copy of the model before continuing to attack the online model.
Inversion Attack	Attacker recovers sensitive information about the training data. This can include full reconstructions of the data, or attributes or properties of the data. This can be a successful attack on its own or can be used to perform other attacks such as Model Evasion.
Prompt Injection Attack	An attacker creates prompts to an LLM that make the LLM act in unintended ways. These "prompt injections" are often crafted in such a way that the model ignores parts of its initial instructions and obeys the attacker's prompts instead.
Traditional Cyber Attack	Attacker uses Tactics, Techniques, and Procedures (TTPs) from the cyber domain to attain their goal. These attacks may target model artifacts, Application Programming Interfaces (API) keys, data servers, or other foundational aspects of AI compute infrastructure distinct from the model itself.

Table 2.1: High-level descriptions of adversarial attacks and their possible effects [10]

2.3 Large Language Model

A LLM is a type of AI program that is able to understand, summarize and generate text [14]. The reason why an LLM is able to interpret human language is that they are trained on millions of gigabytes worth of text gathered from they internet. To understand how words and sentences function together an LLM uses a type of machine learning (ML) called deep learning (DL) where through probabilistic analysis of unstructured data distinctions between pieces of content can be recognized without the intervention of a human.

The DL technique used in most cases is based on transformer architecture, which is a specialized type of neural network [15]. Transformers use a mathematical technique named self-attention, that enables the model to detect relationships between tokens, even when they are far apart in a sequence [14]. These elements are called tokens which are smaller units of text, such as words, subwords, or characters. To help the AI model understand these tokens, they are transformed into embeddings, which are numerical representations (often multi-dimensional vectors) capturing the meaning and relationships

between tokens [15]. The self-attention mechanism is then able to assign a score to each token which is called a weight to determine the similarities, correlations and other dependencies. [16].

The use case of LLM is very broad and ranges across different industries and applications. They enable and improve chatbots and virtual assistants to provide human-like responses in customer service [15]. LLMs are also invaluable in content creation, by automating tasks such as writing blog posts, marketing content and even code. In the academic and research fields, they help summarize large data sets and accelerate knowledge discovery. In addition, LLMs are used for language translations, which is a further step towards breaking down language barriers.

While there are many use cases for LLM, there are still some challenges and limitations. Development and operational costs are high due to the need for expensive hardware and large data sets. LLMs also raise ethical concerns, including biases, privacy issues and the potential to generate harmful content. Another challenge is explainability, as it is often difficult to understand how the model arrives at certain results. In addition, issues such as AI hallucinations, security risks and the complexity of managing billions of parameters make effective troubleshooting and security of LLMs difficult.

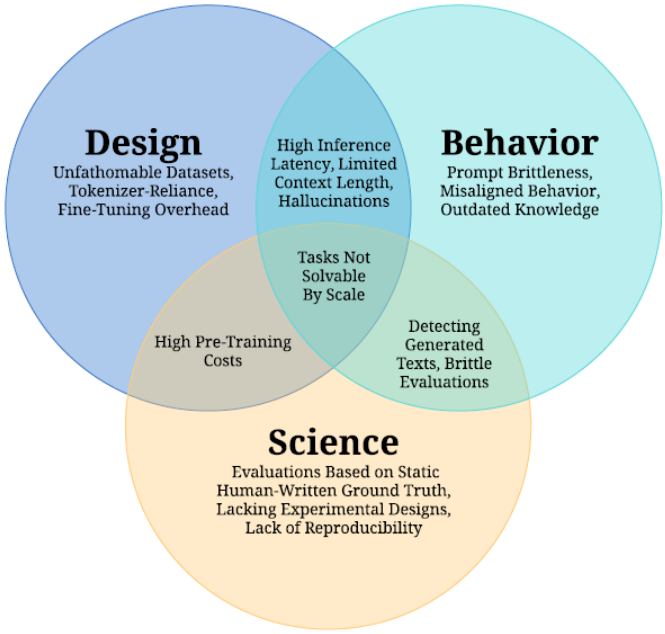


Figure 2.1: Overview of LLM Challenges [17]

Although there are many use cases and benefits to utilizing LLMs, there are also some important risks and challenges with this topic. Some key challenges are described in the paper [17] and shown in Figure 2.1. One of the prominent problems of LLMs is hallucination. LLMs often give the wrong or meaningless information, especially when the query is about something that they have not been trained on. This poses a risk in applications such as medicine or law, where accuracy is critical. Another problem that is worth to mention is the high inference latency. The computational cost and time required for LLMs to generate responses can hinder real-time applications such as chatbots.

2.4 Retrieval Augmented Generation

RAG is a technique for improving LLMs using the integration of external knowledge databases. A fundamental limitation of LLMs is their dependency on fixed training data, which can lead to outdated or incomplete information [18]. In addition, this limitation makes it difficult to respond to requests for specific internal information. To address this limitation, RAG allows you to add relevant KBs that allow adding relevant information to the prompt to provide additional context. The flow of this process is illustrated in Figure 2.2. By bridging the gap between LLMs and dynamic information retrieval, RAG greatly improves the capabilities and reliability of AI systems.

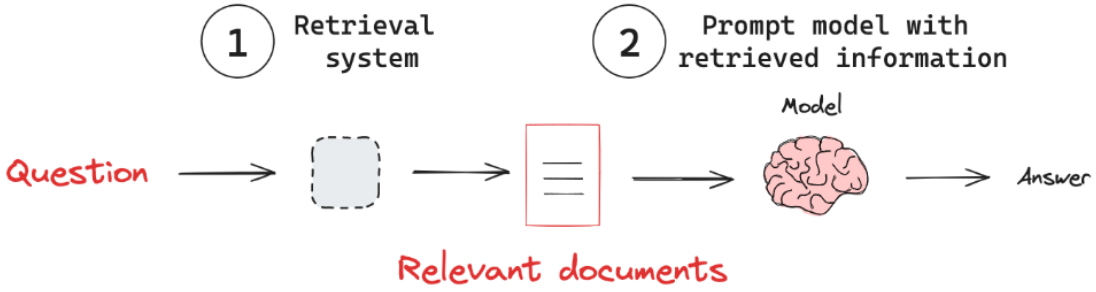


Figure 2.2: Key concept of RAG provided by the paper [18]

Another approach used by companies to address this major limitation of LLM is fine-tuning [19]. This raises the question about the difference between RAG and fine-tuning. Although both methods customize LLMs for specific use cases, they are fundamentally different in their methodology. During fine-tuning, a model is trained with the internal data and therefore the model parameters are trained for this specific task. In contrast to this, RAG does not train the model, but simply uses the data from an external KB to read important information into a model using natural language processing. To summarize, RAG supports prompt engineering by leveraging an internal data to improve the output, while fine-tuning retrains the model on a specific data set to improve the output.

RAG offers several fundamental advantages [20]:

- **Cost-efficient AI implementation and AI scaling**: Cost-efficient AI implementation and scaling: organizations often start with base models that have been trained with public data. Fine-tuning these models for specific domains is expensive and requires a lot of resources. RAG enables organizations to improve the performance of their models by integrating internal, authoritative data without retraining, which reduces costs.
- **Access to current domain-specific data**: Since generative AI models have a knowledge cutoff, their relevance drops over time. RAG mitigates this problem by providing real-time access to data, which improves accuracy and timeliness.
- **Lower risk of AI hallucinations**: Generative AI models generate answers based on learned data patterns, but can provide incorrect information. RAG helps to mitigate this problem by relying on trusted data to improve answer accuracy, although it does not completely eliminate errors.

- **Increased user trust**: AI systems rely on the trust of users. RAG models increase credibility by containing references to external data sources. This allows users to verify information.
- **Expanded use cases**: By integrating retrieval mechanisms, RAG models can access and merge information from different sources, which improves their overall capability and usefulness.
- **Enhanced developer control and model maintenance**: Companies that implement RAG benefit from robust data pipelines and storage solutions that enable developers to adapt KBs for different tasks. This reduces the need for retraining and enables targeted optimization.
- **Greater data security**: Since RAG simply adds the external knowledge with natural language processing and does not use the knowledge as training data as in fine-tuning, a separation between the model and the external knowledge is maintained.

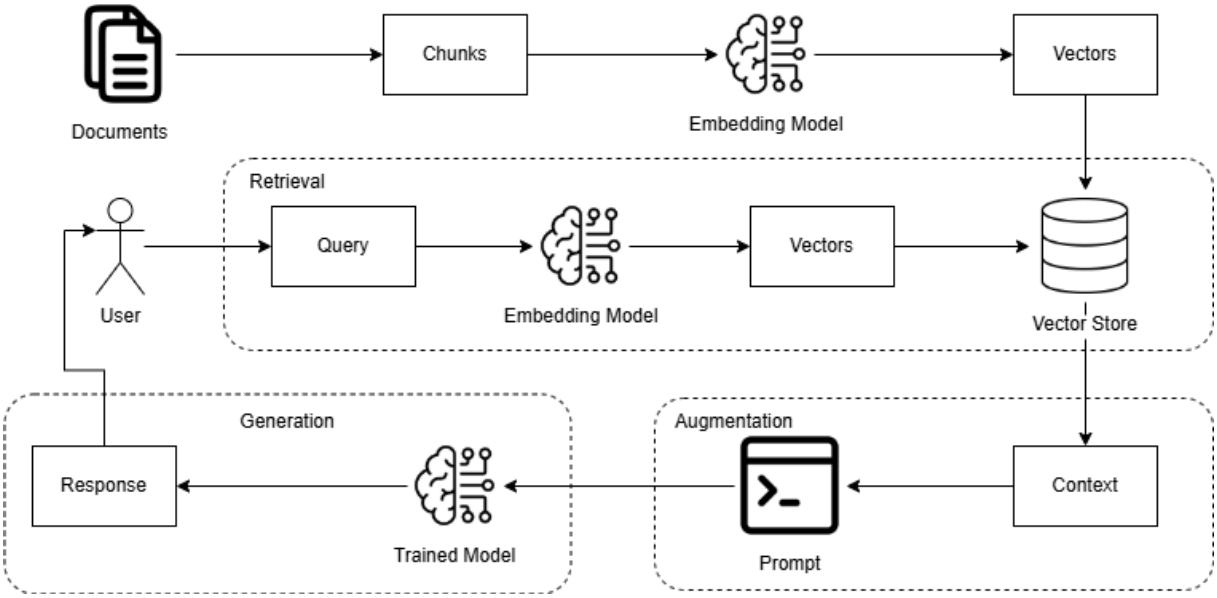


Figure 2.3: Typical RAG Architecture

2.4.1 Chunking

To divide large documents into smaller segments, there is a process called chunking [21]. In the RAG process, these chunks are converted into embeddings, which are then stored in a vector database. An embedding model is used to calculate the embeddings. This process is presented in Figure 2.3. In a RAG system, a semantic search can then be used to retrieve only the most relevant chunks.

Each RAG application has different use cases and data sets, which makes it necessary to choose a suitable chunking strategy. Choosing the right strategy is crucial as it defines the boundaries for chunk creation [21]. According to [21] and [22], these are the most used chunking methods:

- **Fixed-size chunking**: This strategy separates the text into chunks, which all have a certain fixed size. An optimal chunk size is usually determined by experimentation and measured in tokens [21]. Since this method makes no distinction between sentence boundaries or semantic meaning, it is optimal for cases where chunk size has a higher priority than context, e.g. when analyzing large amounts of data from genetic sequences or standardized data sets such as surveys [22]. To ensure that context still exists between the chunks, there is a technique called chunk overlapping. This allows you to select a number of overlapping tokens, which then overlap between neighboring chunks [21]. The advantages of fixed-size chunking include its simplicity, its wide applicability to different data types, its minimal computational requirements, and its independence from ML models or specific linguistic considerations [22]. However, limitations include the loss of semantic context, inflexibility when dealing with variable text structures, and redundancy in overlapping segments. These disadvantages can lead to inaccurate results for complex documents or redundant information.

- **Recursive chunking**: This strategy is similar to fixed-size chunking, but offers a more adaptive solution [22]. Instead of dividing the text at a certain length, the text is split up using several separators, such as paragraph breaks or sentence endings, until an optimal chunk size is reached. The advantage of this method lies in its ability to keep context by achieving a balance between chunk size and context preservation. However, a disadvantage is that the strategy is more complex, because it requires an optimal separation hierarchy. Therefore, it can still break sentences or paragraphs if configured incorrectly, which can lead to semantic loss.

- **Semantic chunking**: An alternative strategy is semantic chunking, where the text is divided in such a way that it is grouped based on the semantic similarity of its embeddings [22]. Semantic chunking is ideal for complex systems where the preservation of context between chunks is important. The advantages of this strategy are its ability to ensure meaningful and context relevant chunks through contextual grouping, effectively manage overlap without losing quality and provide significant application versatility. The disadvantages, however, include greater complexity and computational cost due to the need to embedding models for clustering, as well as a resulting dependency on these models.

- **Document-based chunking**: This chunking strategy separates the text according to elements such as headings, lists, chapters, etc. For this reason, it makes sense to use this strategy for documents that are already well formatted, such as Markdown files [22]. The advantage of this strategy is that the structure and logical flow of the documents can be used in the chunks. However, this requires a well-structured document, which can quickly lead to pre-processing step if the format is not exactly as expected.

2.4.2 Precision and Recall

It might be asked why RAG is used instead of putting all relevant information directly into the LLM itself. The main reason is that LLMs have a limit on how much text can be processed in a single run, known as a context window [23]. GPT-4o, for example, has

a context window of 128K tokens [24]. While this is a considerable number, it is still a limitation. RAG mitigates this limitation by ensuring that the most relevant information is retrieved and passed to the LLM as context, rather than trying to pass all knowledge directly into the model.

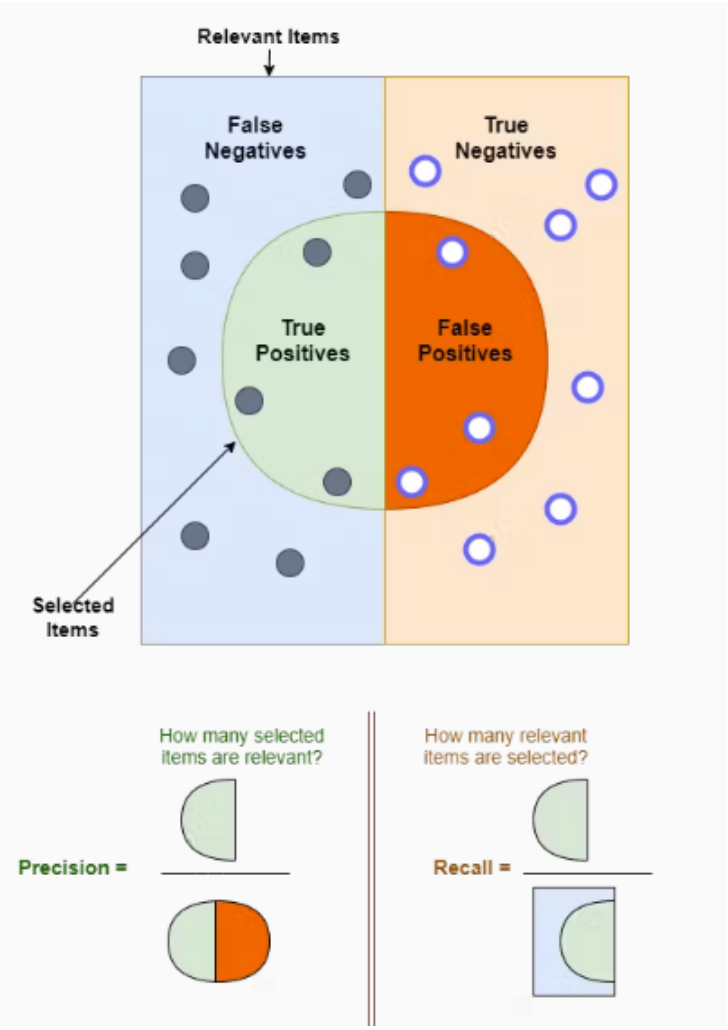


Figure 2.4: Precision vs Recall [25]

To evaluate how effectively RAG finds relevant information with the help of the vector search, two key metrics are usually used: *Precision* and *Recall* [26]. These metrics, which are shown in Figure 2.4, provide information about the retrieval performance. It can be said that precision is the ratio of retrieved relevant elements to all retrieved elements, while recall is the amount of retrieved relevant information in relation to all relevant information.

However, there is a fundamental trade-off between precision and recall. Precision improves as false positives decrease, while recall improves as false negatives decrease [27]. In the context of RAG, this trade-off becomes clear when the number of retrieved documents, often referred to as k, is adjusted. Increasing k improves recall as more potentially relevant documents are retrieved, which reduces the likelihood of missing important information. However, this has a negative impact on precision, as searching for more documents also

increases the likelihood that irrelevant documents will be included, leading to more false positives.

2.4.3 Reranking

In addition, the two metrics Precision and Recall are not rank-dependent, so they do not take into account the position or rank of the retrieved information [26]. However, it was shown in Paper [28] that LLM performance decreases when the position of the relevant information changes, suggesting that the models have difficulties to use information in long context. In particular, performance tends to be lowest when models have to retrieve information from the middle of a long input context. As a result, relevant documents should ideally appear at the beginning or end of the retrieved context, and it should not contain too many false positives.

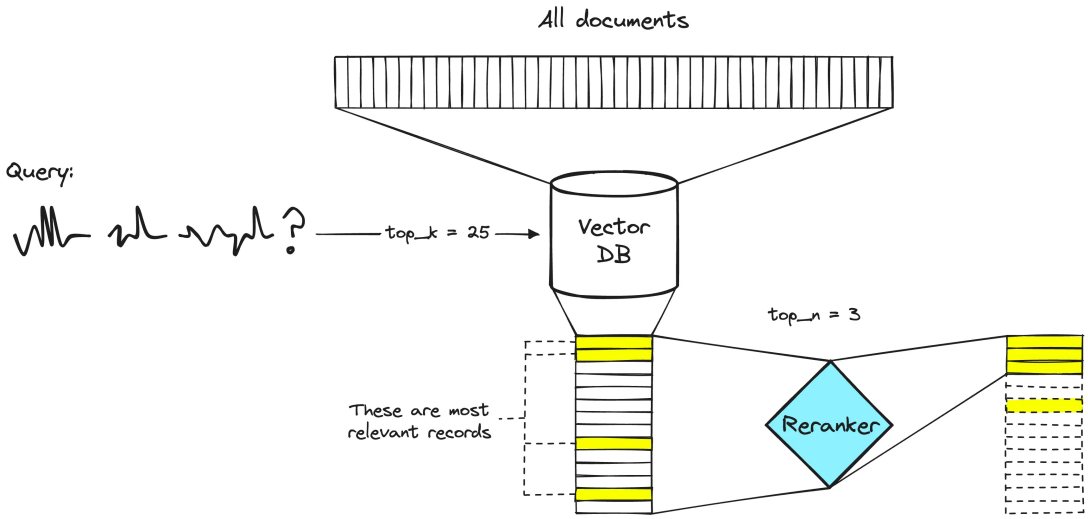


Figure 2.5: Reranking Process based on [23]

This problem is directly addressed by reranking. The inclusion of rerankers improves the precision and relevance of search results for complex information retrieval tasks [29]. Figure 2.5 presents the reranking process graphically, which can be summarized as follows:

1. **Initial Retrieval**: The first step is to find the top-k documents using a similarity-based retrieval method [30].

2. **Semantic Scoring**: The retrieved documents are then evaluated by an LLM-based reranking model that assigns a relevance score to each document based on its semantic match with the search query [30]. This step utilizes the LLM's ability to interpret context and recognize semantic similarities.

3. **Reordering**: Based on the relevance scores assigned in the previous step, the documents are reordered so that the documents with the highest semantic relevance appear first [30].

4. **Final Selection**: In the final selection step, the top k scoring documents are used as the final context, improving the accuracy of the overall RAG system [30].

This reranking approach ensures that the most relevant information is prioritized, mitigating the limitations and improving the overall quality of the retrieval. While reranking improves overall quality, it leads to a trade-off between the benefit of improved performance and the additional computational cost required [31].

Chapter 3

Related Work

LLMs are finding more and more use in our daily lives [32]. It is, therefore, only logical that they are also being used in the field of Information Technology (IT) security, especially for tasks such as threat modeling. Threat modeling is an important area where AI can support stakeholders in identifying and addressing potential risks.

This chapter provides an overview of the current status of research on the topic of using LLMs in threat modeling. The first section outlines the methodology used for the literature review. This is followed by a detailed analysis of the identified literature to present the current research in this environment. Finally, a discussion summarizes the findings and highlights the key limitations of the existing research.

Special attention is given to the key constraints that define the focus and methodology of this work. These constraints include the requirement that the LLM must provide explanations for each detected threat to improve explainability, leveraging a pre-trained LLM without additional fine-tuning for specific areas, and prioritizing of smaller and less complex systems to address scalability challenges.

3.1 Methodology

A literature review was performed to analyze the current state of research on using LLM-based solutions in threat modeling. Multiple academic search engines were used for the review, including Google Scholar, IEEE Xplore, ACM Digital Library, Springer and Elsevier. The following search terms were used to find relevant works:

- “LLMs threat modeling“
- “LLMs in threat analysis“
- “Threat Modeling with AI“
- “LLMs in automated threat modeling“

- “Large Language Models in threat analysis“
- “RAG in Threat Model“
- “Facilitate Threat Modeling by using LLM“
- “AI-assisted threat modeling“

Only papers whose title or abstract referred to the use of LLM in security practices such as threat modeling, threat identification, or vulnerability identification were considered. In addition, studies dealing with the use of AI in threat modeling were also included. However, papers focusing only on AI applications in threat identification were excluded, as these often diverge from threat modeling and address other security-related topics. Similarly, papers that focus on the usage of LLMs in threat intelligence were excluded unless they also address the use of LLMs for vulnerability or threat identification. Studies that dealt with threat models for LLMs were also excluded, as the literature review aimed to evaluate how LLMs can be used in the field of threat modeling and not how threat models can be applied to LLMs.

A total of 16 papers were identified by using this method. The related work sections of these papers were then reviewed to identify additional studies that could address existing research on the use of LLMs in threat modeling. However, no previous research that focuses explicitly on LLMs in threat modeling was mentioned in these sections. Nevertheless, an additional Google search discovered a video and two articles describing experiments on the use of LLMs in threat modeling. These sources were considered relevant and included in the study. For this reason, a total of 19 resources were identified, which are discussed further in the following Section 3.2.

Based on the criteria the [33] fit the inclusion of papers. However, the paper was not accessible until 4 January 2025, and a request remain unanswered. Therefore, despite its potential relevance in this area, this paper is not included in this thesis.

3.2 Literature Overview

This section reviews a collection of 19 resources which includes 16 academic papers, two online articles, and one video, to provide an understanding of the current landscape of AI-powered threat modeling and identification. As shown in Table 3.1, the review is divided into three thematic categories: LLMs in Cybersecurity, AI for Threat Modeling, and LLMs for Threat Modeling, and provides insights into key advances and trends in each area. Out of these categories, LLMs for threat modeling is the central focus of this thesis and, therefore, the most important and most analyzed category. This focus reflects the main goal of the thesis to investigate the capabilities of LLMs in threat modeling methods.

References	Type	Category	Scope	Evaluation
[34]	Paper	Practical	LLMs in Cybersecurity	Model Benchmark
[35]	Paper	Practical	LLMs in Cybersecurity	Dual Dataset
[36]	Paper	Practical	LLMs in Cybersecurity	Cross-Language LLM
[37]	Paper	Practical	LLMs in Cybersecurity	None
[38]	Paper	Practical	LLMs in Cybersecurity	Case Study
[39]	Paper	Practical	LLMs in Cybersecurity	LLM Performance
[40]	Paper	Theoretical	LLMs in Cybersecurity	None
[41]	Paper	Practical	LLMs in Cybersecurity	Accuracy of 42 LLMs
[42]	Paper	Theoretical	AI for Threat Modeling	None
[43]	Paper	Practical	AI for Threat Modeling	Theoretical ML Algorithm
[44]	Paper	Theoretical	AI for Threat Modeling	None
[45]	Paper	Practical	LLMs for Threat Modeling	Performance with 72 Questions
[46]	Paper	Practical	LLMs for Threat Modeling	Experiment
[47]	Paper	Practical	LLMs for Threat Modeling	KG Functionality Evaluation with Competency Questions
[48]	Paper	Theoretical	LLMs for Threat Modeling	None
[49]	Video	Practical	LLMs for Threat Modeling	None
[50]	Paper	Practical	LLMs for Threat Modeling	Performance Experiment
[51]	Online Article	Practical	LLMs for Threat Modeling	None
[52]	Online Article	Practical	LLMs for Threat Modeling	None
This Study	Master Thesis	Practical	LLMs for Threat Modeling	Interview, Performance

Table 3.1: List of References

3.2.1 Large Language Models in Cybersecurity

There are several papers that address the topic of vulnerability detection with LLMs. Paper [34] shows a RAG approach that makes cyberattack investigation better than GPT-4o. The tool achieves this by performing vulnerability identification with complex question about attack strategies and tactics. Paper [35] examines an LLM customized for intrusion detection in satellite networks that achieves remarkable accuracy by processing network traffic similar to text data. Paper [36] presents a framework called LLM4Vuln that is able to detect vulnerabilities in code through logical reasoning and external contextual data such as documentation, API details or system architecture. The evaluation of this framework was performed on real systems on bug bounty platforms, where it uncovered severe vulnerabilities. These papers show how customized LLM solutions can improve vulnerability detection techniques in specific areas.

Papers [37], [38] and [39] are chapters from books that analyze different aspects of the use of LLMs in cybersecurity. Chapter [37] addresses the dual role of LLMs in cybersecurity by discussing both their usefulness in detecting threats and vulnerabilities and the associated

security risks. In addition, practical mitigation techniques for these risks are proposed. Chapter [38] focuses on the use of ChatGPT to create attack trees that visualize potential cyberattack scenarios. The study shows that ChatGPT can help in creating initial drafts of attack trees. Finally, chapter [39] shows how LLMs improve cyber threat hunting by analyzing patterns in log analysis. It supports real-time detection and improving threat intelligence. Combined, these chapters demonstrate both the potential and the challenges of integrating LLMs into cybersecurity practice.

Papers [40] and [41] are use cases that deal with LLM in general cybersecurity. Paper [40] even categorizes five different use case possibilities of LLM in cyber defense: (1) Threat Intelligence, (2) Vulnerability Assessment (Pentest, SAST), (3) Network Security, (4) Privacy Preservation, (5) Operation Automation (Incident Response). The paper [41] evaluated 42 LLMs in various security related task like malware detection, phishing detection, and intrusion detection. These results showed that LLMs will play an increasingly important role in cybersecurity.

3.2.2 Artificial Intelligence for Threat Modeling

The paper [42] contains several ideas on how AI can be used in IT security. On the one hand, the use of LLMs for threat detection is proposed. However, the author rather refers to the use of LLMs in the area of reading log files or emails for phishing detection. In the area of threat modeling, a predictive approach is proposed and the role of LLMs in the ability to predict potential vulnerabilities in the future. Although this paper focuses more on conceptual ideas and less on practical applications, it still shows the possibility of how AI can be used in IT security.

Paper [43] presents a practical ML framework that uses several existing ML algorithms to detect hybrid cyber threats in the Industrial Internet of Things. Existing ML algorithms include *Random Forest*, *Grey Relational Analysis*, and others to detect complex threats in networked industrial systems. The paper focuses very heavily on the theoretical aspects of the algorithms. While the paper provides a structured approach to modeling hybrid cyber threats, it does not do an empirical evaluation of the methods. For this reason, the paper can be considered as an overview of current approaches and as a conceptual framework.

The paper [44] deals with the idea of using AI in threat detection and in IT security in general from a theoretical point of view. The main focus is on how AI can improve threat detection via pattern recognition, anomaly detection, and predictive analytics. Despite focusing on the theoretical views, this paper highlights the potential of AI to defend against emerging threats and the value of AI-driven protection mechanisms in a modern cybersecurity system.

3.2.3 Large Language Models for Threat Modeling

The paper [45] provides a new approach to threat modeling that uses Llama 2 with RAG to identify security threats and provide insights about the system. It focuses on two

References	Type	Practical	Early Stage	Direct Approach	Threat Detection on AI Systems	LLM for Threat Detection	LLM for Threat Validation	DFD Usage
[45] (2024)	paper	yes	no	yes	no	yes	no	no
[46] (2024)	paper	yes	not specified	yes	no	no	yes	yes
[47] (2024)	paper	yes	yes	no	no	yes	no	yes
[48] (2024)	paper	no	not specified	not specified	no	yes	no	no
[49] (2024)	video	yes	yes	yes	no	yes	no	yes
[50] (2023)	paper	yes	not specified	not specified	no	not specified	not specified	not specified
[51] (2023)	online article	yes	yes	no	yes	yes	no	no
[52] (2023)	online article	yes	yes	no	no	yes	yes	no

Table 3.2: Comparison of references in the context of LLMs for threat modeling

key questions from the Threat Modeling Manifesto: MQ1 `What are we working on` and MQ2 `What can go wrong`. The approach provides practical LLM-based support to address these critical aspects. To ensure that the LLM receives sufficient information about the system, the LLM is provided with 3 to 60 PDF pages of documentation. The tool was evaluated using six questions on 12 threat models, with human raters scoring the answers based on how well they met their expectations. The result of the RAG-enhanced model outperforms the LLM base model with a satisfaction rate of 75%. The reason for this is that the RAG model provides more accurate and concise answers, as well as recognizing faster when a question could not be answered due to insufficient information. In conclusion, this paper makes an important contribution to automatic threat identification and system understanding in the context of threat modeling.

A different use case of LLMs in threat modeling is proposed by [46]. It identifies threat validation as a critical challenge in the threat modeling process. The study aims to explore how LLMs can assist users in validating security threats within a system. To this end, an experiment was performed in which participants were divided into four groups, each of which was given different combinations of DFDs, LLMs, and a list of pre-identified threats (a mixture of true and false positives). The results show that LLMs improved threat validation by helping participants recognize threats more accurately, although at the cost of an increased number of false positives. It is notable that the groups that were using LLMs had the lowest technical knowledge, which may have contributed to the higher false positive rates. Although no specific LLM was developed in the study, it demonstrates the potential of integrating LLMs into the threat modeling process to improve validation accuracy.

[47] presents an approach that uses LLMs to convert DFDs into machine-readable knowledge graphs. This method enables the identification of threats within a system. The LLM not only generates the knowledge graph from the DFD, but also performs reasoning tasks to analyze the system's own threats. The approach was evaluated using a series of competency questions. It was found that LLM efficiently generates knowledge graphs and successfully expands the scope of threat identification. These results confirm the practical value of LLMs for facilitating threat modeling.

In contrast to the other three studies, [48] operates on a theoretical level and presents ideas for the implementation of LLMs in four key areas of cybersecurity based on a literature review. The author identifies these areas as (1) Threat Feature Detection, (2) Automated Attack Tree Generation, (3) Data Generation and Preprocessing, and (4) Intrusion and Threat Discovery. Threat modeling is only addressed in the discussion of Threat Feature Detection. Here, the author shows how models such as *RoBERTa* can improve threat identification by showing relationships between attack vectors and defenses and providing a conceptual link to threat modeling.

In a webinar [49], a practical implementation of LLMs in the threat modeling process is presented. In this presentation, the Chief Research Officer outlines a three-step approach to create a suitable threat model.

1. The input consists of a DFD, a markdown file describing the functions of the software, and a prompt. Based on these inputs, the LLM generates security objectives and asset information and outputs them also in the form of a markdown file.

2. The LLM uses the security objectives and asset information along with another prompt to create specific threat scenarios for the software.

3. The LLM uses the threat scenarios, security objectives, and a new prompt to create a strategy to mitigate the identified threats.

This webinar, led by AppSecEngineer's Chief Research Officer, shows how LLMs can effectively support the threat modeling process in practice[49].

[50] is slightly more distant from the topic of LLMs in threat modeling but still closely related. It analyzes how LLMs, such as ChatGPT, can support threat analysis for critical systems. Using experiments, the researchers evaluated ChatGPT's ability to identify potential threats. The study included 78 queries to evaluate feasibility, utility, and scalability. The results showed that ChatGPT provided moderately useful input: 64% of responses were feasible, 35% were useful, and effectiveness decreased with the complexity of the system. This paper is relevant because hazard analysis and threat modeling use similar methods to identify risks, although they focus on safety and security, respectively. The findings from hazard analysis can inform the development of LLM applications for threat modeling.

Several online resources address the use of LLMs in threat modeling and provide practical examples rather than academic insights. Two such examples are highlighted at this point. Article [51] compares GPT-3.5, Claude2, and GPT-4 and shows that while GPT-4 and Claude2 provide higher quality results, they rely heavily on detailed inputs and iterative refinement. The study included assumptions in the prompts, such as using a SAST tool to rule out certain threats, such as SQL injection. Similarly, Article [52] demonstrates the use of ChatGPT in identifying threats to software functions that use LLM. Despite their differences, these examples illustrate the different ways in which LLMs are used in this area.

3.3 Discussion

This section identifies the main limitations in the existing research, outlines the main problem for this thesis, and defines the limitations to provide a clear scope for the proposed approach.

3.3.1 Limitations

In general, there appears to be limited research on how LLM can support the process of threat modeling. However, based on the few studies, the following concrete limitations appear.

L1) Existing approaches are not applicable to early stages of development.

L2) Intermediate structures are required to effectively detect threats.

L3) Limited experimental work has been conducted on threat identification with minimal input data.

L4) No research exploring how LLMs can be utilized for threat identification in AI systems.

One important limitation is the lack of scientific papers that focus on threat modeling at an early stage, which is the key principle in the Threat Modeling Manifesto [7]. An example of this is the paper [45], which requires 3-60 PDF pages of documentation to detect the threats to a system. However, such system documentation is not available at an early stage of software development. Minimal inputs such as DFDs are often overlooked, although they are a common type of abstraction at this stage.

The second limitation is the dependency on intermediary structures, such as knowledge graphs, created by the LLMs. While these methods led to success in [47], they also introduce another layer of complexity. For this reason, there is a research gap in detecting the threats directly from DFD via LLM, making the reasoning similar to the practical webinar [49].

In this context, there is a lack of experiments that show how well an LLM performs threat identification with minimal input, like a DFD. The lack of such experiments limits the understanding if LLMs are able to create a comprehensive threat model in the early stages.

The L1-L3 limitations become even more important when looking at the limitations within a specific domain. In this case, the field of AI security highlights these limitations, as there is currently little to no research investigating how LLMs can be used to detect threats specifically related to AI systems. This is a significant research gap (L4), as AI systems often come with their own specific threats that differ from traditional software threats. The focus on AI-specific threats reinforces the problem, as the existing research does not even cover general gaps in this area.

3.3.2 Main Opportunity and Constraints

Based on the identified research gaps, there is an opportunity to develop an LLM-based approach that enables threat modeling for AI architectures during the early design phase of software development. This approach would rely on minimal input, such as DFDs and brief feature descriptions, and avoid intermediary structures by directly generating the threat model from the DFD using the LLM. As seen in Table 3.2, no existing paper fully meets all conditions. Therefore, this master thesis takes the opportunity to fill these gaps by proposing an approach that fulfills all the conditions visualized in Table 3.2. By addressing these gaps, this thesis aims to demonstrate the practicality and value of direct, early identification of threats through LLMs in AI systems.

To differentiate this thesis from existing work, additional requirements were defined. Some of these constraints were already defined before conducting the literature review: the requirement that the LLM must provide explanations for each detected threat to improve interpretability, the use of a pre-trained LLM without fine-tuning, and the priority on smaller and less complex systems to address scalability challenges. However, further limitations could be identified from the literature review.

These constraints include the restriction to limited documentation that limits input to a minimum, such as DFDs and short functional descriptions, which reflects the reality of early software development. This limitation opposes the approach in paper [45], which requires extensive documentation of 3 to 60 pages. The constraint that there are no intermediate structures highlights a direct approach in which the LLM identifies threats without relying on intermediate structures. This prioritizes simplicity while potentially reducing precision. This approach is in contrast to paper [47], which relies on intermediate structures such as knowledge graphs. Finally, the two-stage threat identification and validation constraint defines a process in which threats are first identified and then validated in a second stage to ensure accuracy and validation capability. This constraint was heavily influenced by the paper [46].

These constraints are fundamental to the approach of this thesis as they define the unique scope of the work and, at the same time, present limitations that could serve as directions for future research.

Chapter 4

Architecture and Prototypical Implementation

This chapter provides a detailed insight into the architecture and implementation of a prototype that can be used to identify and validate threats with minimal inputs. First, the four main components of the system and their interactions are described, with a focus on the integration of a RAG system to extend the capabilities. Key technologies such as Docker, Flask, Ollama, ChromaDB and React are introduced and their role in the process is explained at the same time.

The chapter then continues with a detailed implementation documentation of the ThreatFinderAI component and describes the functions, the changes and the role of the component in identifying and validating threats. It also covers the Flask Backend with the main topics of its architecture, key endpoints, DFD parsing, prompt engineering and integration with the ThreatFinderAI.

In the end, there will be an understanding of the structure of the prototype, the rationale for its design, and the technologies used to implement it.

4.1 Architecture

To show how the developed prototype was designed to identify and validate threats with minimal input, the architecture chapter plays a crucial role. This chapter provides an overview of the components of the system, their interactions, and the basic structure of the prototype. In addition, it also illustrates how the prototypical architecture represents a RAG system. This section aims to provide a clear understanding of how the system is built and how its parts interact to achieve its goals.

4.1.1 High-Level Architecture

As shown in Figure 4.1, the prototype divided into four main components: ThreatFinderAI, Flask Backend, LLM, and ChromaDB. The ThreatFinderAI component acts as the frontend of the prototype and was developed prior to this work [3]. However, significant changes and additions were required to allow dynamic inputs, such as a created DFD, to be effectively processed by the LLM. In order for an LLM approach to be properly integrated into the ThreatFinderAI tool, three additional components were implemented for the prototype. The Flask Backend acts as a simple API that coordinates the requests from the frontend appropriately. It is therefore also responsible for ensuring that the requests are rightly forwarded to the LLM and the ChromaDB component in a correct sequence. The LLM component, which is using the Ollama technology as a default, enables a local execution of modern LLMs and ensures that sensitive threat model data remains on the user's client and is not passed on to third parties. The Llama 3.2 model is used in this prototype. In addition to generating outputs, the LLM component is also used to calculate embeddings. The ChromaDB component complements these functions by storing the embedding vectors. ChromaDB as a vector database and technology is described in more detail in Section 4.2.

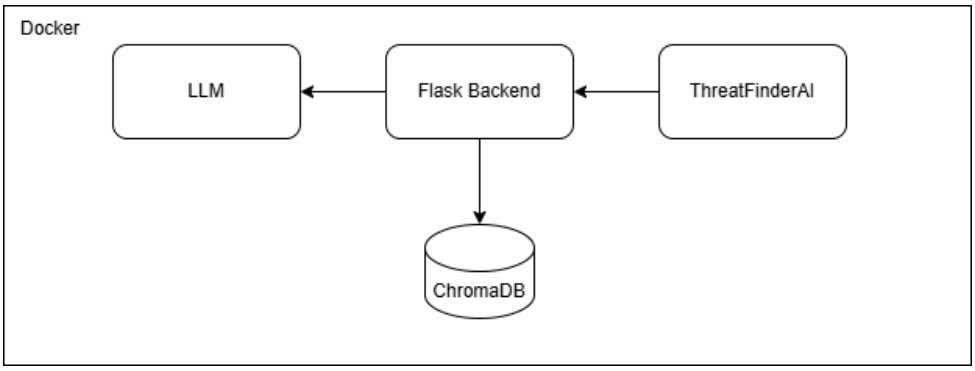


Figure 4.1: High-Level Architecture

The typical workflow starts with the Flask Backend receiving input files and splitting them into chunks of text. These chunks are then processed by the LLM component to obtain an embedding for each chunk. These embeddings are then stored in the ChromaDB component. Once this process is done, the ThreatFinderAI can identify and validate threats with the corresponding request to the Flask Backend component. Upon threat identification, the backend retrieves the relevant threat embeddings in ChromaDB based on the DFD input it received from the frontend. These found threats are then sent back to the frontend and in case of a threat validation, the backend sends these threats together with a prompt to the LLM component. The detailed mechanism of the backend processing and communication is discussed in Section 4.4, while Section 4.1.2 provides an in-depth analysis of the technologies and the RAG process.

All four components are provided as separate Docker containers, which ensures a simplified and uncomplicated setup process, as shown in Listing 4.1. Docker Compose is used to initialize the system and automates the configuration of the components. The

ThreatFinderAI frontend operates on port 3000, the Flask Backend on port 5000, Ollama on port 11434, and ChromaDB on port 8000. The Ollama setup includes a custom bash script called entrypoint.sh that is used during setup to download the Llama 3.2 model. Persistent volumes are configured for ChromaDB and Ollama to ensure that embeddings and the LLM model are maintained across container restarts. This reduces the time needed for restarting as the LLM does not need to be downloaded again nor the embeddings recalculated, which makes the full process more efficient.

Listing 4.1: Docker Compose Configuration File for Basic Infrastructure Setup

```
version: "3.8"
services:
  chromadb:
    image: chromadb/chroma:latest
    container_name: chromadb
    ports:
      - "8000:8000"
    volumes:
      - chromadb_data:/data

  frontend:
    build:
      context: ./react-frontend
      dockerfile: Dockerfile
    container_name: frontend
    ports:
      - "3000:3000"
    environment:
      - REACT_APP_BACKEND_URL=http://127.0.0.1:5000

  ollama:
    image: ollama/ollama:latest
    ports:
        - 11434:11434
    volumes:
        - ./ollama/ollama:/root/.ollama
        - ./entrypoint.sh:/entrypoint.sh
    container_name: ollama
    pull_policy: always
    tty: true
    restart: always
    entrypoint: ["/usr/bin/bash", "/entrypoint.sh"]

  flask-backend:
    build:
      context: ./flask-backend
      dockerfile: Dockerfile
    container_name: flask-backend
    ports:
      - "5000:5000"
    environment:
      - CHROMADB_HOST=chromadb
      - CHROMADB_PORT=8000
      - OLLAMA_URL=http://ollama:11434
      - OLLAMA_MODEL=llama3.2
    depends_on:
      - chromadb
      - ollama
    command: ["sh", "-c", "sleep 5 && flask run --host=0.0.0.0"]

volumes:
  chromadb_data:
  ollama_data:
```

4.1.2 Retrieval Augmented Generation Architecture

The RAG framework is a focus of this thesis, as the entire backend architecture is based on it. This section explains how exactly the RAG architecture was built for the prototype and the technologies used to do it. Figure 4.2 shows the specific RAG architecture implemented in this thesis, while Figure 2.3 shows a typical RAG architecture.

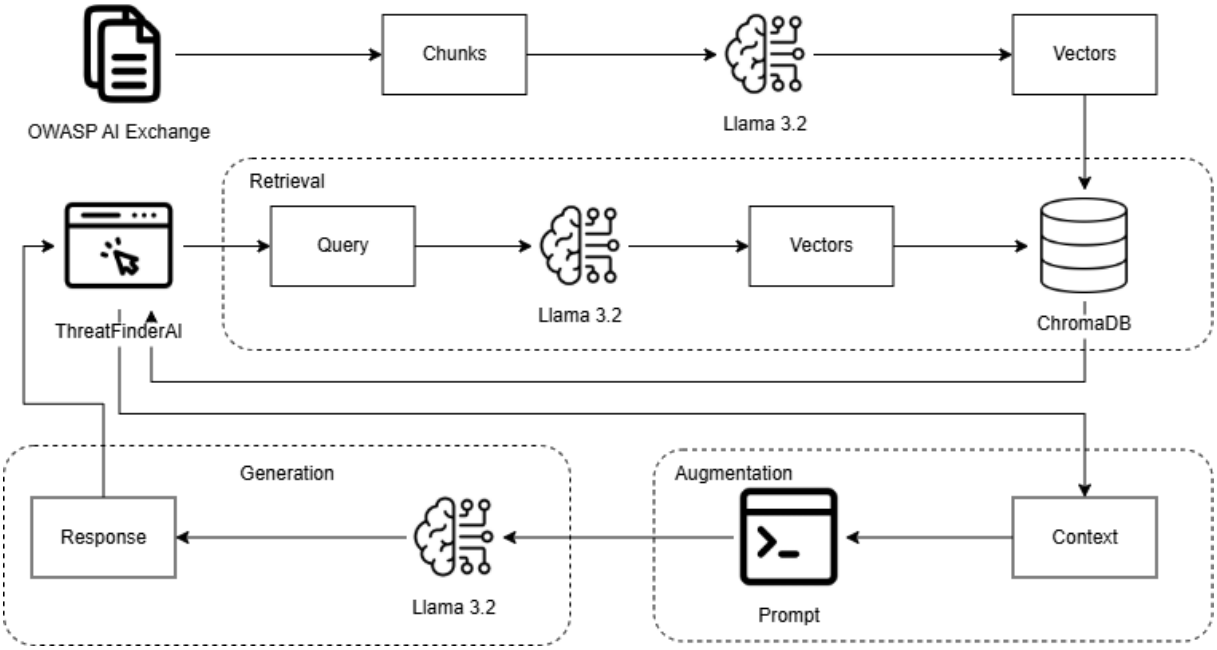


Figure 4.2: Prototype RAG Architecture

To ensure that the LLM has the relevant data on AI threats, all information is collected from [53]. This means gathering information from specific sources to enable the LLM to react appropriately without training it or adjusting its parameters or inner workings. The extracted pages are then split into multiple chunks to ensure in the case of this prototype that each AI threat is treated as an own entity along with its description. The technology used for the chunking is LangChain [54] while the chosen chunking strategy is *document chunking*. In addition, the embedding model Llama 3.2 is used in Ollama to convert these text blocks into vectors. In an earlier version of this prototype, the nomic-embed-text model [55] has also been explored. However, this model has a context length limitation of 2048 tokens. Therefore, the chunking strategy would have to be adapted accordingly. As document chunking was used and a further chunking strategy would have required investigation, the focus was only placed on document chunking. Therefore, the embedding model Llama 3.2 in Ollama was chosen as it does not have this limitation. The technology used for storing the vectors is ChromaDB [56], which is widely recognized for its efficiency in processing high-dimensional embeddings.

The second part of the RAG system is responsible for the threat identification process. As shown in Figure 2.3, a user is visualized in the RAG process, which represents the user interface of ThreatFinderAI in the prototype system. This representation includes the creation of the DFD. This DFD is then processed in the backend by splitting it into separate assets, which are shown as queries in the figure. Each asset is then converted

into vectors using the same embedding model as before. After that, a similarity search on ChromaDB is performed to find the most relevant AI threats. This process represents the retrieval step.

Comparing Figure 2.3 with Figure 4.2, a significant difference to the traditional RAG process is recognizable. In the prototype developed for this thesis, the retrieved results are sent back to the ThreatFinderAI as shown in Figure 4.2. This diversion ensures that threat identification and threat validation run as separate processes, which is the two-stage process discussed in Section 3.3. Once the user triggers threat validation, the identified threats are inserted back into the standard RAG process, initiating the augmentation step.

The following augmentation step begins with the identified threats, which are the output of the similarity search. This output is used as a context for the LLM and is therefore combined with a prompt and sent to the LLM, which in this prototype is Llama 3.2 running on Ollama. However, the output may change depending on the LLM used. Therefore, different LLMs are evaluated and compared in Chapter 5. In the final generation step, the retrieved threats are formatted into a structured output using the LLM and sent back as a response to the ThreatFinderAI user interface, where the user can review the threats identified by the prototype.

4.2 Technology

This section gives an overview of the technologies that were used to develop the prototype. Thereby, the reasons for their use as well as the places where the technologies were used are explained.

4.2.1 Docker

Docker is a technology that allows applications to be separated from the underlying infrastructure, which enables a faster deployment [57]. This effect is achieved by running the applications in so-called *containers*, which can be described as isolated environments. This results in the advantage that several containers can run in the same host, but are independent of each other and can therefore ensure security and consistency. Furthermore, not all required dependencies have to be packed on one system, but are also separated from each other. This allows the user to run a prototype quickly while avoiding dependencies problems on the host system.

As shown in Figure 4.1, each block in the high-level architecture represents a separate container. To simplify the setup process, Docker Compose is used for container deployments. The corresponding *docker-compose.yml* file, which is shown in Listing 4.1, defines four services and two volumes that ensure persistent data storage.

The use of volumes is important because they avoid time-consuming reinstallation processes when restarting containers. This is important for the following services:

1. **Ollama**: This service contains an *entrypoint.sh* script that downloads the required LLM model at the start. Without a volume, the model would have to be downloaded again each time the container is restarted, which would lead to unnecessary time delays.

2. **ChromaDB**: This component is responsible for saving the embeddings correctly so that they can be used later for threat identification (see subection 4.2.4). Since calculating and saving these vectors can be time-consuming, a volume is used to ensure that the data remains persistent even after restarting the container.

It can therefore be said that volumes with its persistent data storage increases efficiency and also enhances the user experience.

4.2.2 Flask

To ensure that the ThreatFinderAI tool is able to effectively interact with an LLM, a simple API was developed to act as an interface between ThreatFinderAI and the LLM. This minimal API is using Flask [58], which is a lightweight web framework for building web applications in Python. The term "lightweight" and "micro" in Flask stands for the goal of keeping the core framework simple while being highly extensible. In addition, Flask offers a good documentation and builds on two important dependencies: *Jinja2* and *Werkzeug*. *Jinja2* is the template engine used by Flask, while *Werkzeug* provides important tools for implementing a Web Server Gateway Interface (WSGI) application.

The prototype relies on key dependencies, which are all listed in file *requirements.txt* on Github [59]. The Langchain dependency is the most important, as this framework greatly simplifies the development of applications based on LLMs. Langchain offers open-source components and third-party integrations, such as LangGraph, which enables the creation of stateful agents. For production, Langchain provides tools such as LangSmith to inspect, monitor and evaluate applications for optimization. Deployment is simplified by the LangGraph platform, which transforms applications into production-ready APIs and assistants. The prototype uses essential libraries, including *langchain-core* for foundational abstractions, *langchain-chroma* for vector storage and retrieval, *langchain-ollama* for model integrations, and *langchain-text-splitters* for text chunking. To summarize, langchain is a tool that simplifies the LLM-based application development and is therefore very valuable for developers creating such applications.

4.2.3 Ollama

Ollama is an open source framework that allows users to run LLMs locally [60]. This is done by combining model weights, configuration and data into a single package. In addition, Ollama optimizes setup and configuration, including Graphics Processing Unit (GPU) usage. The framework allows the use of multiple models and offers users a wide range of options. One of the more popular open source models available in Ollama is Meta's Llama 3 which is a family of models [61]. Ollama has different versions of these

models. For example, the Ollama 3.2 model includes versions with parameter sizes of one billion, which require around 1.3 GB, and 3 billion, which require around 2.0 GB [62]. In the prototype, Ollama runs in a seprated Docker container and only interacts with the Flask Backend. The reason for using Ollama is that LLMs can be run locally. This ensures that the data never leaves the defined infrastructure, which enhances security.

4.2.4 ChromaDB

Chroma is an open source AI vector database designed for building LLM applications [63]. It enables efficient storage of embeddings and their metadata as well as fast vector search, which makes it a strong candidate for semantic search and RAG applications. While Chroma was primarily designed for Python and JavaScript/TypeScript client Software Developement Kids (SDKs), other developers and organizations have developed clients for other languages. Although Chroma is open source, it is important to note that Chroma is also a company that develops and maintains the open source project of the same name. Nevertheless, according to [63], they committed to the principle that every function that is useful for an individual developer remains 100% open. In the prototype, ChromaDB operates like Ollama in a separate Docker container and interacts exclusively with the Flask Backend. The reason for using ChromaDB is that it is very straightforward to set up and open source. It can also be hosted locally, which allows hosting the whole prototype on your system.

4.2.5 React

Unlike the other technologies, React JS is used for the frontend component, which is called ThreatFinderAI. As the ThreatFinderAI tool was developed before this thesis, its use was previously determined, so a decision-making process was not required for the frontend part. React is a library designed for creating user interfaces and best known for its React DOM features, such as components and hooks that work within the browser's DOM [64]. The advantage of using a virtual JavaScript DOM is the higher performance and better scalability of the application.

4.3 ThreatFinderAI

The ThreatFinderAI component is unique in the prototype because it is the only component that existed before this thesis. This section therefore first shows what the original functionality of this component was. Then it highlights the architectural changes that were made as part of this thesis. These changes are then described in more detail, as they have influenced the threat identification and validation. For this reason, key code changes and design decisions that enabled these features are also outlined in this section.

4.3.1 Existing Functionality of ThreatFinderAI

The ThreatFinderAI tool is designed to support and automate threat modeling for AI-based systems by systematically identifying assets, threats and countermeasures while quantifying residual risks [3]. It takes a structured approach similar to traditional threat modelling tools, such as the Microsoft Threat Modelling Tool or CAIRIS, but does in addition address the specific challenges posed by AI threats. This is achieved through an integrated KB of AI-specific vulnerabilities, structured diagrams for modelling system components and automatic risk quantification using Monte Carlo simulations.

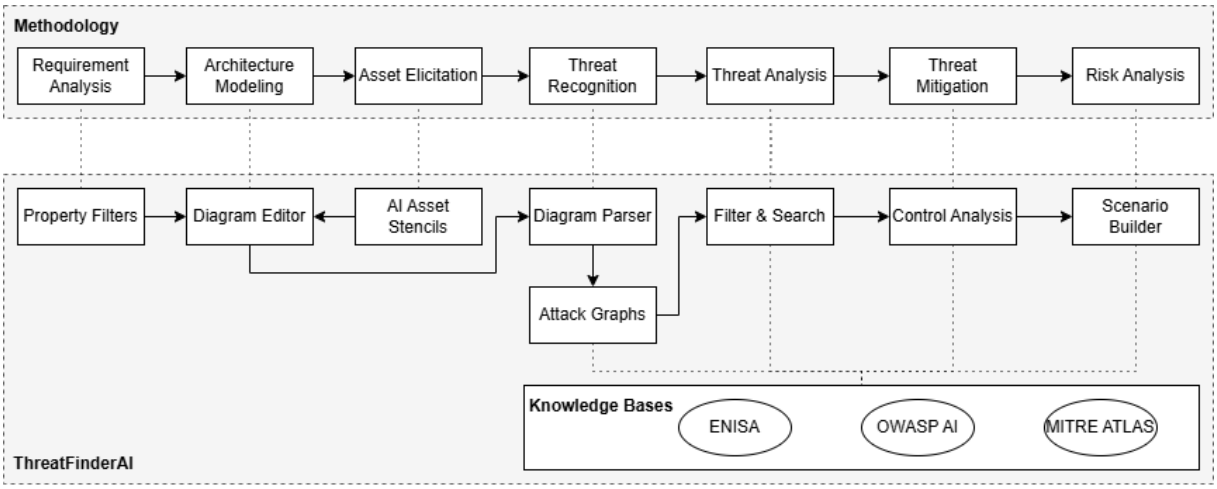


Figure 4.3: Previous Version of the ThreatFinderAI Design

As shown in Figure 4.3, the previous version of ThreatFinderAI already followed a structured process before the development of the prototype in this thesis. The process is described in detail in the paper [3]. For the purpose of completeness, a brief overview is provided below:

- **Step 1**: In the first step, during the requirement analysis, the scope of a business mission had to be defined together with an important security requirement and an asset. For example, it is possible to identify data assets and confidentiality as important security properties as part of the CIA (Confidentiality, Integrity, and Availability) triad.
- **Step 2**: The second step was architecture modeling, where it is important to create a high-level overview of the system. The AI lifecycle model can be used as a guide to ensure that all activities and systems involved were identified in the DFD.
- **Step 3**: In the third step, asset elicitation, the annotations were retrieved from the architecture diagram and mapped to a metamodel that defines the assets within the diagram.
- **Step 4**: In the fourth step, threat recognition, the identified goals were used as input. The ThreatFinderAI tool identified the threats using an attack graph that queried multiple KBs that were matched by a graph model. Specifically, the catalogues from

ENISA [65], OWASP AI Exchange [53] and MITRE ATLAS [10] were converted into a graph-based structure. These sources were then linked to the metamodel based on properties such as asset category and lifecycle stage, which enabled automatic identification of threats.

- **Step 5**: In the fifth step, threat analysis, users have to review the identified threats and add relevant threats to the threat model. Threats directly related to the previously defined key object or security objective are highlighted to prioritize the analysis.

- **Step 6**: The sixth step covers the identification of mitigation controls where users can investigate technical, organizational or strategic countermeasures. This step is automated by querying the KBs, using the meta-model to match the threats with the relevant mitigation strategies based on their lifecycle stage.

- **Step 7**: Finally, in a risk analysis step, previous threat models can be reused for strategic risk analysis to gain additional insights, which means that it generates value through collaboration.

4.3.2 Modifications with Proposed Prototype

To ensure that ThreatFinderAI could be used effectively in the prototype of this thesis, various changes to the existing architecture were required. In Figure 4.4, the elements highlighted in blue show the changes that were made to the existing ThreatFinderAI tool. In specific, the methodologies for threat recognition and analysis have been updated. However, it is important to note that the original modules remains unchanged. The new process, shown in Figure 4.4, has been introduced as an additional option for identifying and analyzing threats, rather than replacing the previous methods. This means that the traditional approach to threat identification can still be used, but there is now an additional option for identifying and analyzing threats using an LLM.

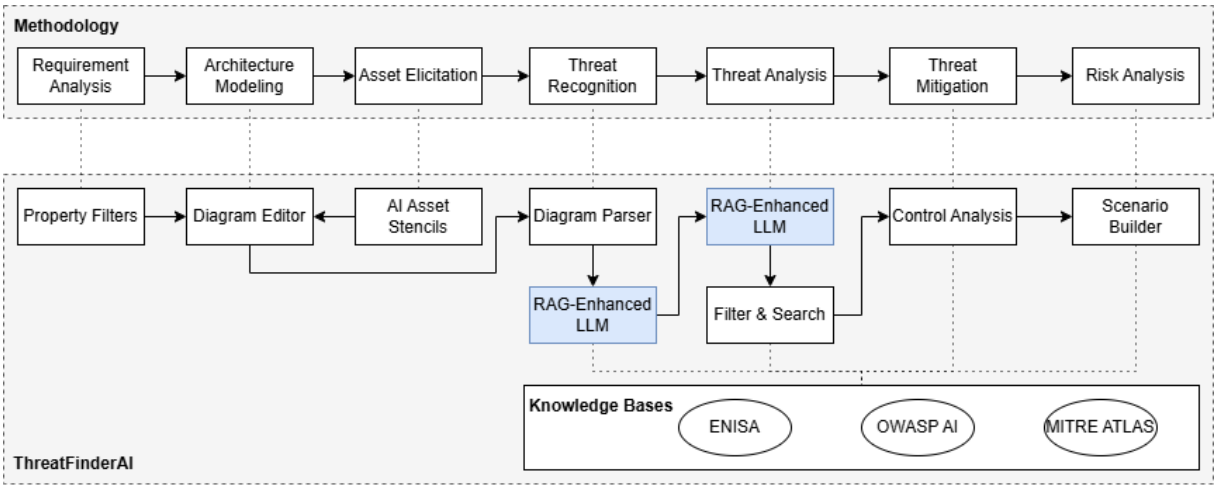


Figure 4.4: Adapted ThreatFinderAI Design – Novel Components in Blue

As shown in Figure 4.4, the updated ThreatFinderAI tool now has two separate requests sent to the Flask Backend: threat identification and threat validation. More details on these requests can be found in Section 4.4. However, it can be said that the ThreatFinderAI has been modified to allow these two requests to be sent to the backend with the DFD in the request body. The motivation for this separation of the two requests is mainly that these two requests should be evaluated separately from each other. For this reason, the identification of threats with minimal input and the validation of threats with minimal input are tested separately in the evaluation Chapter 5.

It is also important to note that no changes have been made to the asset definitions. This means that the LLM can only process asset categories defined in Paper [65], which is based on the generic AI lifecycle reference model. These assets are specific to AI systems and include models, procedures and data, all of which can be compromised or corrupted through intentional or unintentional causes.

4.3.3 Threat Identification

Three new states were introduced to implement threat identification: *threatsFromBackend*, *isLoading* and *selectedKBValue*. The selectedKBValue state is by default set to *enisa* and should trigger logic that depends on this value. The *isLoading* state was introduced to provide visual feedback in the frontend while the request in the backend is processed. This is because the communication with the LLM can take longer than with standard requests. Finally, the *threatsFromBackend* state saves the threats identified by the LLM and enables further processing.

The most important change was the addition of the *handleKBChange* function, which is triggered whenever the KB selection changes. When *llm* is selected as a parameter, then the function triggers a POST request to the "threat_detection" endpoint in the backend. This request contains a DFD diagram, which is retrieved from the local storage in an Extensible Markup Language (XML) format. While the request is being processed, the *isLoading* state is set to true to give users a visual feedback. Once the backend response is received, the identified threats are stored in the *threatsFromBackend* state.

The *getThreatsForCategory* function has been modified to ensure that the threats are displayed correctly. If the threatsFromBackend list contains at least one element and *selectedKBValue* is set to *llm*, the function filters the threats so that only those with names that match at least one threat identified by the backend are displayed. The *getThreatsForCategory* function has been modified to ensure that the threats are displayed correctly. If the *threatsFromBackend* list contains at least one element and *selectedKBValue* is set to *llm*, the function filters the threats so that only those with names that match at least one threat identified by the backend are displayed. This behaviour can be seen in the Listing 4.2. The *getThreatsForCategory* function is always executed when the threat list is rendered in the frontend. The filter mechanism checks whether the threat name contains the string that matches the threats found by the backend.

This decision ensures that even if the LLM returns a threat name such as *Prompt Injection* that does not have an exact match in the predefined threat taxonomy, related threats such

as *Direct Prompt Injection* and *Indirect Prompt Injection* are displayed. This is because it checks whether the string *Direct Prompt Injection* contains the substring *Prompt Injection*. This approach gives attention to threats that are classified as important by the LLM, even if they are not classified with full precision.

Listing 4.2: Filtering Threats based on the Function getThreatsforCategory

```
function getThreatsforCategory(category) {
    if (isThreatValidation===true){
      var threats = threatTaxonomyLLM.filter(t => t['Affected
          assets'].includes(category));
    }
    else{
      var threats = threatTaxonomy.filter(t => t['Affected assets'].includes(category));
    }
    if (threatsFromBackend.length > 0 && selectedKBValue==="llm") {
      threats = threats.filter(t => {
        return threatsFromBackend.some(response =>
            t["Threat"].toLowerCase().includes(response.toLowerCase()));
      });
    }
    threats.map(t => {
      t.potentialKeyThreat = t['Potential Impact'].includes(selectedModelInfo.keyProp);
      return t;
    });
    return threats;
}
```

4.3.4 Threat Validation

Once threat identification is complete and the selectedKBValue remains to *llm*, users can trigger the threat validation. The goal was to enable a POST request for the threat validation inside the threat categories. However, a problem occurred because the threat descriptions and other metadata retrieved from the taxonomy were only available in the threat list, which is displayed separately for each category. This would have led to multiple duplicated threat validation requests, which would have increased processing time.

To solve this problem, a new state, *isThreatValidation*, has been implemented. When the button below the KB selection is clicked, this status is set to true, and a new taxonomy, called threatTaxonomyLLM, is created within the handleThreatValidation function. This taxonomy only contains threats in the state threatsFromBackend. A POST request is then sent to the backend, which updates the descriptions accordingly. The exact implementation details of the function handleThreatValidation can be seen in Listing 4.3.

Listing 4.3: Source Code of the handleThreatValidation Function

```
function handleThreatValidation() {
    setIsThreatValidation(true)
    var threatTaxonomyLLM = threatTaxonomy.filter(threat =>
      threatsFromBackend.some(backendThreat =>
        threat.Threat.toLowerCase().includes(backendThreat.toLowerCase())
      )
    );
    setThreatTaxonomyLLM(threatTaxonomyLLM)

    const postThreatTaxonomy = async () => {
      ...
    }
    postThreatTaxonomy();
}
```

The *getThreatsForCategory* function is triggered as soon as the request is completed. It checks whether the state isThreatValidation is still true, and if so, it uses the threatTaxonomyLLM list instead of the default threat taxonomy. This behavior can be seen in Listing 4.2. The state of isThreatValidation is changed back to false when the handleKBChange function is triggered by switching to a different KB.

To summarize, the threat validation process has been implemented by the handleThreatValidation function, which primarily creates the *threatTaxonomyLLM* list and sends the corresponding POST request to the backend. In addition, a condition has been added to getThreatsForCategory function to check whether isThreatValidation is true. If this is the case, the new ThreatTaxonomyLLM list is used, otherwise the default ThreatTaxonomy list is applied.

4.4 Flask Backend

This section analyzes the Flask Backend component in detail. It starts with an overview of the component-based architecture and describes the key components of the Flask Backend and their roles. After that, it introduces the endpoints provided by the API and shows how the communication with the different modules and functions in the backend works. The third section covers the implementation of the DFD parser and explains how an XML-formatted DFD is processed and transformed into structured components that contain relevant information that is then used to create a system description. This is followed by a discussion of the prompt engineering methods used in this prototype to communicate with the LLM. The final sections look at the backend implementation of threat identification and validation, and they describe in more depth how these methods were implemented in the backend and what its contributions are.

4.4.1 Component-Based Architecture and Tasks

The Flask component is the most important element of the prototype, as it acts as the central node for the management of all communication within the system. As an API, it receives requests from the ThreatFinderAI frontend and handles them accordingly. With that, the Flask application is organized into five components as shown in Figure 4.5: *App.py, DFDParser.py, Chunker.py, ChromaLoader.py and PromptHandler.py.*

1. **App.py**: This is the main component of the backend, which is responsible for handling five endpoints that process incoming requests. Each endpoint corresponds to a specific feature of the prototype and enables a communication between the frontend and the backend.

2. **DFDParser.py**: This component contains all the functions required to interact with the DFD. It can use raw XML files to extract relevant content and convert it into text descriptions. These text descriptions are crucial for providing the LLM an understanding of the system structure.

3. **Chunker.py**: The *Chunker.py* component implements the chunking strategy for document data. Large text inputs are split into smaller chunks to ensure efficient handling and storage.

4. **ChromaLoader.py**: *ChromaLoader* manages all communication with the ChromaDB component. This component is the vector database used to store and query embeddings. These interactions include storing chunks, performing similarity searches and creating new collections within the database. *ChromaLoader* also communicates directly with Ollama. Since the creation of collections requires a predefined embedding function, *ChromaLoader* connects to Ollama directly to compute embeddings. This integration simplifies the embedding workflow by consolidating the process into one component.

5. **PromptHandler.py**: The *PromptHandler* also communicates with the Ollama component, but has a different main task. It focuses on creating dynamic prompts based on the input data. These prompts are then sent to Ollama, while another task of *PromptHandler* is to process the output. As it can be clearly observed, the communication with Ollama is not reduced to a single component, as such a separation would lead to unnecessary back and forth communication between the imaginary Ollama component, *ChromaLoader* and *PromptHandler*. This integration ensures efficient handling of both prompt generation and text generation by the LLM.

To summarize, the backend design efficiently divides responsibilities between the individual components and ensures a clear separation of tasks. The interaction with ChromaDB and the handling of DFD and documents are well organized in specific components. However, communication with Ollama is shared between the *PromptHandler* and ChromaLoader.

4.4.2 Endpoints

This section lists all the endpoints shown in Figure 4.5, explains their respective tasks and describes the process that takes place when an endpoint is called.

Upload Endpoint

The upload endpoint ensures that documents are stored in ChromaDB so that the RAG system can access relevant information about AI threats. This endpoint is designed to handle POST requests and requires a body parameter with the key *file* that accepts one or more Markdown files. After receiving the files, as shown in Figure 4.5, the endpoint processes them in the App.py component. The files are read and sent to the chunker component to use the *mdDocumentChunker* function to split the documents into smaller chunks.

After the text has been processed into chunks, these chunks are sent to *ChromaLoader* component, in which the text chunks are converted into embeddings by the function *addDocumentsToVectorstore*. This function interacts with Ollama's embedding model,

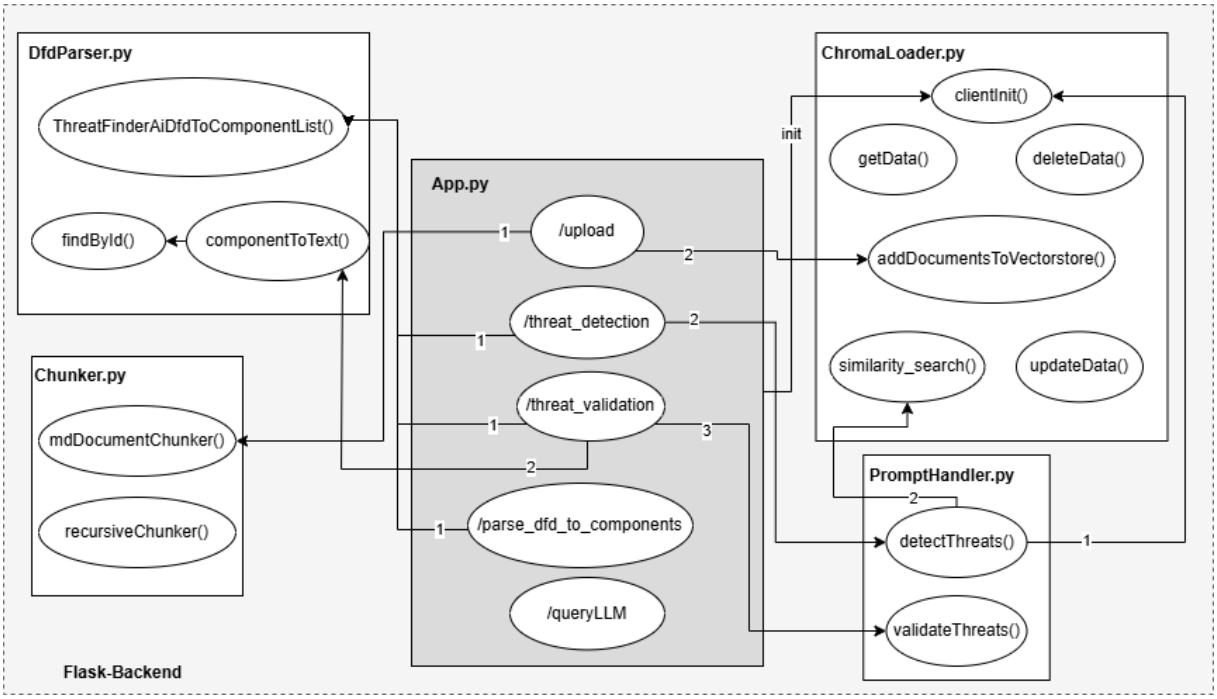


Figure 4.5: Flask Backend

where after the calculation the chunks are stored in ChromaDB, in which they can be retrieved for future queries.

Threat Detection Endpoint

The threat detection endpoint is used to identify threats based on a provided DFD as input. It functions as a POST request, where the DFD must be delivered as an XML file under the key *xml* in the JavaScript Object Notation (JSON) body. As described in Figure 4.5, this endpoint uses multiple backend components to transform the raw XML data into useful content that is then used to identify threats.

When a request is received, the XML content is first forwarded to the *DFDParser* component. Within this component, the *ThreatFinderAiDfdToComponentList* function is used to process the XML to extract all relevant assets described in the DFD. This operation is important as it converts the raw data of the diagram into a structured list of system assets. These extracted components serve as the basis for the following threat identification process.

In a second step, the list is then sent to the *PromptHandler* module, where the *detectThreats* function starts the identification of potential threats. To be able to do this, the *PromptHandler* creates a connection to ChromaDB via the *ChromaLoader* component. The connection gets initiated by the *ClientInit* function, which ensures that the corresponding collection in ChromaDB is accessed, which contains the embeddings for AI threats. Using the extracted DFD assets as input, the system performs a similarity search in ChromaDB with the *similarity_search* function. This search identifies the stored threats that are most relevant to the assets described in the DFD.

Threat Validation Endpoint

Based on the reasoning presented in Section 3, the threat identification and threat validation processes have been designed as separate functionalities, although they could also be combined. The threat validation endpoint is implemented as a separate component whose main goal is to provide better descriptions for the detected threats in relation to the given DFD. This endpoint also works as a POST request and requires two parameters in the request body, which are transmitted as a JSON object.

The first parameter with the label *dfd* corresponds to the format used in the Threat identification Endpoint. It consists of a JSON object with an XML representation of the DFD under the key *xml*. The second parameter labeled *threats* is a list of the identified threats that need to be validated. As shown in Figure 4.5, the process begins with the extraction of the relevant system components from the DFD. This is done by the usage of the *ThreatFinderAiDfdToComponentList* function within the *DFDParser* module, which ensures that the assets and associated data are identified correctly. Once the components are extracted, the *DFDParser* module converts the DFD information into a text description using the *componentToText* function. This description summarizes the key features of the system in a textual format that can be used as input for the following validation step.

With this text about the system, the process continues in the *PromptHandler* module. Here, the ValidateThreat function uses the system description and a dynamically generated prompt (details in Section 4.4.4) to interact with the LLM in Ollama. The LLM generates the threat descriptions by modifying them to the specific context of the system. In addition, the LLM adds a ranking to each threat and provides an estimation of how important the threat is for this system and the arguments in this way. Finally, the generated output is reviewed to ensure its quality before it is returned to the frontend. In addition, the ranking created by the LLM is included in the threat description to increase the depth of analysis. This process is designed to ensure that raw threat identifications can be transformed into robust and custom threat reports that fit the system.

Optional Endpoints

When looking at Figure 4.5, it is clear that there are two additional endpoints. Like the other endpoints, the *parse_dfd_to_components* endpoint is also a POST request in which the DFD must be provided as an XML under the *xml* key in the JSON body. However, the use case for this endpoint is limited as it does not fulfill any essential function within the tool. Its main purpose is to debug the *ThreatFinderAiToComponentList* function. However, with this endpoint, the backend can create a detailed list of all components of a specific DFD XML file.

Another endpoint that is primarily used for debugging is the *queryLLM* endpoint. This endpoint is also a POST request and requires the key *prompt* in the request body. The value related to this key is the question that can be forwarded directly to the LLM. This allows the user to communicate directly with the LLM by making requests and getting responses without additional handling.

4.4.3 Data-Flow Diagram Parser

As shown in Figure 4.5, the DFD parser is a main component of the Flask Backend. Its main function is the conversion of DFDs in XML format into a structured list of components. This list contains all relevant elements with their key attributes that are required for the system description and threat identification.

Component Identification in XML-Formatted DFDs

To achieve a smooth conversion, a system was developed that allows assets to be recognized and information to be extracted. The challenge is that the components consist of a different number of XML tags. For example, a *trust boundary* consists of six tags (three *mxCell* and three *mxGeometry* elements), while an *adversary* component has only two (one *mxCell* and one *mxGeometry*).

To facilitate the identification of components, a predefined list of component types has been created to make it easier to identify the DFD elements. This list, which is shown in Listing 4.4, defines how the DFD parser identifies the components. The *detectionType* key specifies whether the identification should focus on an XML tag or an attribute. The key *detectionKey* determines which key is to be examined, while *detectionValue* specifies the expected value that defines the component type. This method ensures the correct identification of each component generated by the ThreatFinderAI tool.

An important parameter in this identification process is *processingCount*, which specifies the number of entries connected to a specific component. This helps the DFD parser to determine when it should start detecting a new component. It is important to mention that Component recognition must take place at the first entry and is limited to this tag or attribute.

Listing 4.4: Detecting Components based on the Component Type List

```
componentTypes = [
    {"type": "trust boundary", "processingCount": 6, "detectionType": "attrib", "
        detectionKey": "style", "detectionValue": "group"},
    {"type": "bidirectional arrow", "processingCount": 7, "detectionType": "attrib", "
        detectionKey": "style", "detectionValue": "endArrow=classic;startArrow=classic
        "},
    {"type": "undirectional arrow", "processingCount": 7, "detectionType": "attrib", "
        detectionKey": "style", "detectionValue": "endArrow=classic"},
    {"type": "arrow", "processingCount": 2, "detectionType": "attrib", "detectionKey": "
        edge", "detectionValue": "1"},
    {"type": "adversary", "processingCount": 2, "detectionType": "attrib", "detectionKey
        ": "style", "detectionValue": "shape=image;verticalLabelPosition=bottom;
        labelBackgroundColor=default;verticalAlign=top;aspect=fixed;imageAspect=0;image=
        data:image/svg+xml,PHN2ZyB4bWxu...;"},
    {"type": "note", "processingCount": 2, "detectionType": "attrib", "detectionKey": "
        style", "detectionValue": "text"},
    {"type": "asset", "processingCount": 3, "detectionType": "tag", "detectionValue": "
        object"},
]
```

Component Information Extraction from XML-Formatted DFDs However, it is not enough to simply identify the components, the extraction of relevant metadata for the system description is just as important. In the following bullet list, the components are broken down, and it is explained how the information is extracted and what function it should fulfill:

- **Trust Boundary**: The first entry with the *mxCell* tag provides the *id*, which is essential to determine which elements belong to a specific trust boundary. The second entry with the name *mxGeometry* extracts the position and size of the trust boundary. Finally, the fifth entry provides the *value* that is used as the label for the trust boundary.

- **Arrows**: All arrows take their main attributes from the first entry with the tag *mxCell*. The most important elements are *source* and *target*, which specify the *id* of the components that the arrow connects.

- **Adversary**: The first entry contains the *parent* attribute, which specifies the *id* of the corresponding trust boundary. The second entry contains the position and size of the adversary component.

- **Note**: Similar to the Adversary component, the first entry identifies the *parent* part, while the second entry extracts the size and position.

- **Asset**: The asset component is the most important because most of the components that are created in the ThreatFinderAI tool belong to this category. The first entry collects the *id*, *label* and *assetname* of the asset. The second entry identifies the *parent*, and the third entry extracts the position and size of the asset.

However, there is one notable exception to the data extraction. The trust boundary contains multiple *id* tags for its six entries. Only the first *id* tag is included because it represents the ID of the bounding frame. This is a crucial point because other components refer to this *id* as their *parent* attribute.

Transformation of system components into descriptive text

After parsing, the result is a detailed list of all DFD components, each with its main attributes. This structured list is used as the foundation for both threat identification and validation:

- **Threat Identification**: Only the components of the type *asset* are taken into account. Potential threats are detected on the basis of these components.

- **Threat Validation**: All components that are identified are used to create a textual system description. This description is created by using attributes such as *assetname*, *label*, *source*, *target*, *parent*, size and position. The exact implementation is shown in Listing 4.5.

The process begins with describing Trust Boundaries, which are stored in a list. This list is then used to determine the parent component for other items, such as assets, notes, and adversaries, to identify whether they fall inside a specific trust boundary. If no trust boundary is assigned, the position is checked by using the position and size of the trust boundary to determine whether the component is definitely not within a trust boundary. If neither of the two conditions is met, the component is rated as being outside a trust boundary, which is then expressed in the textual description.

For arrows, the *findById* method is used to find the component connected to the source and target *id*. If it is found, the *label* of the corresponding component is inserted into the text. For the Adversary component, which has no *label*, the *type* is used instead. This behavior can be clearly seen in Listing 4.5.

With this approach, the DFDParser provides a structured way to translate XML-formatted DFDs into actionable data for threat modeling and system analysis.

Listing 4.5: Creating a System Description based on the ComponentToText Function

```
def componentToText(components):
    result = ""
    tbList = []
    for component in components:
        if component["type"] == "trust boundary":
            tb = {"id": component["id"], "value": component["value"], "x":
                component["x"], "y": component["y"], "width": component["width"],
                "height": component["height"]}
            tbList.append(tb)
    tbValues = [tb["value"] for tb in tbList]
    tbString = f"The system consists of {len(tbList)} trust boundaries called: {',
        '.join(tbValues)}. "
    result = result + tbString
    for component in components:
        if component["type"] == "asset":
            tbFound=False
            assetStr = f"There is an asset {component['assetname']} called
                {component['label']}"
            for tb in tbList:
                if component["parent"] == tb["id"]:
                    assetStr = assetStr + f" which is in the trust boundary
                        {tb['value']}"
                    tbFound = True
                    break
            if not tbFound:
                if float(component["x"]) >= float(tb["x"]) and float(component["x"]) <=
                    (float(tb["x"]) + float(tb["width"])):
                    if float(component["y"]) >= float(tb["y"]) and
                        float(component["y"]) <= (float(tb["y"]) + float(tb["height"])):
                        assetStr = assetStr + f" which is in the trust boundary
                            {tb['value']}"
                        break
            assetStr = assetStr + ". "
            result = result + assetStr
        elif component["type"] == "arrow":
            sourceAsset = findById(components, component["source"])
            targetAsset = findById(components, component["target"])
            if sourceAsset is not None and targetAsset is not None:
                sourceLabel = sourceAsset.get("label", sourceAsset.get("type",
                    "Unknown"))
                targetLabel = targetAsset.get("label", targetAsset.get("type",
                    "Unknown"))
                arrowStr = f"The asset {sourceLabel} points to asset {targetLabel} with
                    an arrow. "
                result = result + arrowStr
        elif component["type"] == "bidirectional arrow":
            ...
        elif component["type"] == "adversary":
            assetStr = f"There is an adversary in the drawing"
            for tb in tbList:
                if component["parent"] == tb["id"]:
                    assetStr = assetStr + f" which is in the trust boundary
                        {tb['value']}"
            assetStr = assetStr + ". "
            result = result + assetStr
        elif component["type"] == "note":
            noteStr = f"There is an note in the drawing which says:
                {component['value']}. "
            for tb in tbList:
                if component["parent"] == tb["id"]:
                    noteStr = noteStr + f"The note is in the trust boundary
                        {tb['value']}. "
            result = result + noteStr
    return result
```

4.4.4 Prompt Engineering

There are numerous prompt engineering techniques that are used today to interact with LLM. Some popular examples are zero-shot and few-shot prompting, which uses examples to guide the model to produce the desired output [66]. Another popular method is chain-of-thought prompting, which uses complex reasoning capabilities by introducing intermediate steps in the reasoning process [67]. There is also meta-prompting, an advanced prompting technique that aims to encourage a more abstract and structured form of interaction with LLMs.

For the two main functions, threat identification and threat validation, prompts are used to interact with the LLM. These prompts are structured in such a way that they correspond to the meta-prompting technique. According to [68], meta-prompting is defined by five main characteristics

- Structure-oriented: The focus is on presenting problems and solutions in a general structure rather than specific content.
- Syntax-focused: Use syntax as a guide framework, with a focus on the form and structure of responses as templates for expected outcomes.
- Abstract examples: These examples show the framework of problems and solutions without going into specific details.
- Versatile: Designed to be applicable in different areas and to provide answers for a wide range of tasks.
- The focus is on the logical organization and clear classification of information.

Since this RAG system processes dynamic input, Meta-Prompting is appropriate due to its structure-orientated focus, which clearly defines tasks and focuses on consistent formatting. In addition, the syntax-focused design provides the JSON output requirement, and by using abstract examples with placeholders, the prompt dynamically integrates specific system details while maintaining its general JSON structure. Using this method in combination with the categorical approach, the threat validation prompt ensures that the examples highlight the dynamic integration of names and elements from the system description and increase focus on the dynamic input.

Meta-prompting provides multiple advantages over few-shot Prompting, as it focuses on structure-oriented guidance rather than relying on specific examples [66]. This approach reduces the use of tokens as it favors general frameworks over detailed content, which is therefore more efficient. It also ensures fairer comparisons between models by minimizing the impact of example-specific biases. In addition, meta-prompting is more tied to zero-shot techniques, where the influence of specific examples is minimized. These advantages, combined with its versatility and structural effect, make meta-prompting particularly effective for dynamic applications.

4.4.5 Threat Identification

As explained in Section 4.1.2, the prototype extracts only threats from the OWASP AI Exchange [53] website if no other reports are added to the ChromaDB. A list of these threats can be found below. The threats are further explained either in the OWASP AI Exchange report [53] or in the threat list created for evaluation purposes in the Appendix B.

1. **Threats through use**
   - 1.1. Evasion
     - 1.1.1. Closed-box evasion
     - 1.1.2. Open-box evasion
     - 1.1.3. Evasion after data poisoning
   - 1.2. *Prompt injection*
     - 1.2.1. Direct prompt injection
     - 1.2.2. Indirect prompt injection
   - 1.3. *Sensitive data disclosure through use*
     - 1.3.1. Sensitive data output from model
     - 1.3.2. Model inversion and Membership inference
   - 1.4. Model theft through use
   - 1.5. Failure or malfunction of AI-specific elements through use
2. **Development-time threats**
   - 2.1. Broad model poisoning development-time
     - 2.1.1. Data poisoning
     - 2.1.2. Development-environment model poisoning
     - 2.1.3. Supply-chain model poisoning
   - 2.2. *Sensitive data leak development-time*
     - 2.2.1. Development-time data leak
     - 2.2.2. Model theft through development-time model parameter leak
     - 2.2.3. Source code/configuration leak
3. **Runtime application security threats**
   - 3.1. *Non AI-specific application security threats*
   - 3.2. Runtime model poisoning (manipulating the model itself or its input/output logic)
   - 3.3. Direct runtime model theft
   - 3.4. Insecure output handling
   - 3.5. Leak sensitive input data

It is worth noting that AI threats can be categorized into three main types: Threats through use, Development-time threats, and Runtime application security threats. These categories are highlighted in bold in the list. The classification is based on whether the threat occurs through normal interaction with an AI model, during the training phase, or as an attack on the model/infrastructure at runtime. Threats that are in italics in the list cannot be detected by the tool at the moment. The reason for this is that they are not included in the threat taxonomy, either because they are high-level threats or because they are not specifically relevant to AI. Therefore, they were also not included in the taxonomy of the ThreatFinderAI tool, which was created prior to this thesis and is explained in more detail in the Section 4.3.

As described in Section 4.1.2, the tool uses a document-based chunking strategy to detect threats. This means that each threat is given a chunk that contains the text. The related

headers are added as metadata during the chunking process. As a result, each chunk consists of descriptive text that is converted into embeddings together with the metadata.

For each object identified by the DFD parser, as explained in Section 4.4.3, a similarity search is performed to determine a predefined number (k) of similar texts. The idea behind this approach is that the embeddings for threat descriptions should be calculated in such a way that they come as close as possible to the embeddings of relevant assets. The optimal value of k is evaluated together with the most effective chunking strategy and the input format for the similarity search in Chapter 5.

It is important that the metadata is used in the similarity search output as it contains the chunking headers of the documents that allow the threat to be correctly identified. Each similarity search result is merged into a list that is that includes only threats and excludes unrelated metadata. This finalized list of threats is then delivered to the ThreatFinderAI tool where it is compared to the threat taxonomy to define the final threats. The details of this process inside the ThreatFinderAI tool are further explained in the Section 4.3.

4.4.6 Threat Validation

The threat validation approach combines elements of DFD parser implementation (Section 4.4.3) and prompt engineering (Section 4.4.4). In contrast to threat identification, the threat validation process is not built on the RAG principle. Instead, the threats (identified by the threat detection process - Section 4.4.5) are validated on the basis of the system description generated by the DFD parser and a more detailed and structured prompt.

The threat validation process makes two important contributions:

1. It improves the description of each threat so that it is more specific to the system description. It also explains why or where a particular threat could be exploited in the system.

2. An attempt is made to classify the threats according to their significance for the system. To this end, the LLM assigns a ranking to each threat, where 1 is the most critical threat, followed by lower priority threats in descending order. This is then included in the threat description.

Since threat validation is dependent on the LLM output, there is the potential for errors in these two aspects. The main reason for this weakness is that all threats are sent to the LLM for validation at the same time. This processing can lead to unintended results despite detailed prompt instructions. To mitigate this, the system has been designed to handle errors: In the event of a serious back-end failure, the front-end retains the default threat descriptions so that the process can be easily restarted.

One way to significantly reduce errors in threat validation would be to analyze each threat individually based on the system description. This method was tested, but led to

major disadvantages. Firstly, it significantly increased the processing time. Secondly, the rating system became completely unreliable, as the LLM rated each threat in isolation and therefore could not take into account the relative importance of the threats. Even if previously ranked threats were dynamically fed back into the process, the LLM would have to re-rank all threats at each iteration to correctly adjust the ranking, which further increased the processing time.

As a result, this initial prototype prioritizes speed over minimizing error frequency. The main objective of this thesis is to evaluate the feasibility of validating and detecting threats with minimal input data. The focus is on demonstrating whether this is even possible, and not on building a system for immediate production use.

Chapter 5

Evaluation

This thesis implements a prototype that is able to identify and validate threats in AI systems with minimal input. The architecture and the implementation of this prototype are described in Chapter 4. To evaluate the effectiveness of the prototype, it is important to evaluate its performance qualitatively and quantitatively. This chapter focuses on how this evaluation is carried out and what the evaluation results are.

It is worth noting that there is a basis prototype architecture, which is described in Chapter 4. This prototype utilizes a document chunking strategy as a default within the RAG system. This strategy uses a similarity search approach for each asset, where the default prototype sets the depth parameter k to two. This prototype forms the basis for this thesis. However, for the evaluation, several new configurations and methods are tested to investigate how different approaches perform with minimal inputs.

The first section of this chapter describes the methodology used for the evaluation. Both the objectives and the approach chosen for this chapter are discussed. The second section presents the results of the evaluation, followed by a detailed discussion of the results in the final section.

5.1 Methodology

This section shows the methodology for the evaluation. First, the objectives of the evaluation are discussed and then the specific descriptions of the approaches used.

5.1.1 Objectives

A number of defined objectives were set for the evaluation of the prototype. These objectives aim to evaluate the prototype in both threat identification and validation as the prototype adjusts to different configurations and evaluates which approaches work best. The following objectives have been set for this evaluation:

1. **Expert View**: The gain of insights from experts in different IT areas on threat identification and validation using fictional DFDs.
2. **Critical Threat Prioritization**: The identification and ranking of the most critical threats for two DFDs based on expert input, including their evaluation of LLM-generated threat descriptions.
3. **Best Configuration**: Another goal is to compare system configurations based on expert insights in threat identification.
4. **LLM Effectiveness**: In addition, the determination of the effectiveness of LLMs in generating accurate and actionable threat descriptions through expert validation is also important.
5. **Qualitative and Quantitative Assessment**: The assessment of threat identification and validation using both qualitative expert feedback and quantitative performance analyses to measure system effectiveness.

Taking these objectives into account, the evaluation provides a comprehensive understanding of the strengths and limitations of the prototype. By combining expert interviews and performance analyses in both phases (*i.e.*, threat identification and validation), the research question of whether an LLM-based approach can identify and validate threats to AI systems with minimal input will be answered. During this process, the best practices and configurations for the RAG system will also be identified and documented.

5.1.2 Metrics

Two evaluation approaches are used to identify and validate threats, which are described in Section 5.1.3. Although they differ in terms of methods, both approaches include qualitative and quantitative measurements.

Qualitative Metrics: The qualitative evaluation is based on interviews with four experts from different IT areas. The participants are:

- A senior security expert with broad experience in cybersecurity.
- An AI specialist who also has broad experience in the security sector.
- A software engineer who works as a consultant and acquired basic knowledge of AI during his studies.
- An economist with a degree in digital strategies and years of experience in applying technical knowledge in a business context.

The reason for selecting a diverse group of participants is to capture varied perspectives on AI-related security threats, assuming that people from different areas will look at AI threats differently. For example, a business strategist might focus on economic risks and

challenges, while a security expert might focus on data protection and regulatory issues. However, it is assumed that certain threats will be recognized by all groups together. These threats, which are identified as critical by several participants, serve as the basis for further qualitative analysis of the prototype configurations, which are further explained in Section 5.1.3.

Quantitative Metrics: The quantitative evaluation focuses on the performance of the configurations. The most important metrics are:

- Hardware utilization: Measures the system resource consumption.
- Response time: Evaluates the processing speed of a configuration.
- Error rate: Shows the robustness and reliability of new configuration in the prototype.

These metrics are used to analyze the various configurations.

5.1.3 Approach

The approach chosen for the evaluation varies in the different key phases of the system and is customized to the specific requirements of each phase. These approaches are described in detail to provide a complete picture of how each phase is evaluated.

The first stage is concerned with Threat Identification, that includes the following steps:

1. The first step considers preparation and execution of interviews, where a threat list and two fictional scenarios were created, each accompanied by a DFD. The threat list contained threats that the prototype should be able to recognize, as well as a brief explanation and an example for each threat. This threat list can be viewed at Appendix B and the two fictional scenarios as well as the DFDs at Appendix C. In addition, the interviewees were asked to use this list to identify threats that are relevant to the respective DFD and scenario.
2. In a second step, the interview was interpreted. Once all interviewees had completed the threat identification task, their results were analyzed. The aim was to determine an expected threat state for each DFD and scenario. The threats that were identified most frequently by interviewees were considered the most relevant for the scenario and were used as the optimal identification baseline.
3. As a third step, the various configuration options were examined. Firstly, similarity search strategies were tested because the identification grade is highly dependent on it. There are two configurations that were tested and compared. The first approach involves a similarity search for each asset using a *low-depth k*, which is also the default setting in the prototype. The second approach performs a similarity search for the system description using a *high-depth k*. The main difference between these strategies lies in the way how the threats are identified. In the *low-depth k* strategy,

each asset looks for k threats during the similarity search. Therefore, k may not be large, as the selected threats are assigned to each asset individually. However, these threats are stored in a common list to ensure that each individual threat is only recorded once. In the high-depth k strategy, the input for the similarity search is the system description as a whole, and the threats are identified based on this DFD mapping.

Secondly, chunking strategies were tested. The first and simplest strategy is the document-chunking strategy without any pre-processing. In this approach, the data is loaded directly into ChromaDB in its raw form, keeping the original structure and content, which means it is the *default*. The second strategy involves *manual pre-processing*, where only the information describing the threat is kept, and all redundant content is removed. This ensures that the text focuses exclusively on the relevant details of the threat. The third strategy involves document-chunking with *recursive chunking*, a method supported by resources such as [22], which recommends the use of fixed token sizes per chunk as best practice. For this evaluation, a chunk size of 512 tokens with an overlap of 100 tokens was used, which is mentioned as best practice in [22]. Finally, the fourth strategy involves document-chunking with a *summarization* to optimize both the chunk size and the quality of threat descriptions. In this approach, an LLM is integrated into the chunking process to create these summaries with a chunk size of 1000 tokens, which is also mentioned as best practice in [22]. This strategy is visually illustrated in Figure 5.1. Each of these strategies will be evaluated to determine their effectiveness in improving the identification of threats in the RAG system.

Lastly, a final configuration evaluated in this study involves the implementation of a reranking step as described in Section 2.4. For this evaluation, the best chunking strategy and the best similarity search approach were selected and compared both with and without the reranking step. As a technology, the reranker of Cohere is integrated with LangChain [69]. The aim of this analysis was to determine whether the inclusion of reranking increases the accuracy and effectiveness of threat identification, particularly when working with minimal input data. This comparison should provide an understanding of the benefits of reranking in optimizing the overall performance of RAG systems for threat identification tasks.

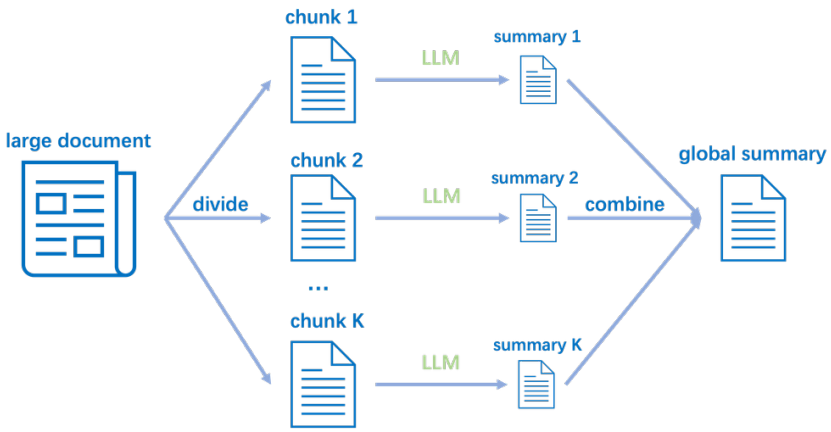


Figure 5.1: Document based Chunking with Summarization [22]

4. The final step is the comparison and the analysis of the results obtained in steps 1 to 3 are compared qualitatively by comparing the threats identified in the interviews with those identified in 9 configurations of the prototype. The main objective of this comparison is to determine which prototype configuration is able to identify threats with minimal input and, as a result, best matches the established baseline for expected threat identification. For the quantitative performance, the time efficiency for the configurations is evaluated. This is the only quantitative metric since the error rate is 0% as threat identification does not depend on an LLM outcome compared to threat validation.

The second stage focuses on Threat Validation, which involves the following steps:

1. As a first step, the prototype configuration was evaluated because in contrast to Threat Identification in Threat Validation, the prototype was tested with several LLMs. The LLMs used include the default *Llama 3.2 with 3B* parameters, as well as *Deepseekr1 (1.5B), Deepseekr1 (7B), Mistral (7B), and Llama 3.1 (8B).* These models were all self-hosted using Ollama which can be seen in the Chapter 4. As an exception to all these self-hosted LLMs, *Gemini 1.5 Flash* was included in the tests to evaluate the qualitative and quantitative differences between self-hosted models and a cloud-based model. The purpose of the comparisons is to evaluate the performance and quality of the two approaches.

2. In a second step, the interviews were prepared and conducted. A threat was selected for both scenarios, and the six LLMs had to validate the threat for the system. Each model provided a detailed description of the threats, including why and where it presents a risk, as well as the possible actions an attacker could take to exploit it. Participants had to rate the quality of each threat description using a Likert scale from 1 to 5, with 1 indicating poor quality and 5 indicating high quality. The evaluation criteria focused on how well the descriptions clarified two important aspects: firstly, where the threat is relevant, and secondly, what the threat means. In addition to the Likert scale, participants also provided qualitative feedback on the evaluation criteria. Dieses Feedback konnte sowohl die Stärken der Beschreibungen sowie auch die areas for improvement gut aufzeigen. This feedback highlighted both the strengths of the descriptions and the areas for improvement.

3. In a third step, the interpretation of the interviews were conducted. The qualitative criteria were combined to analyze the trends in LLM output based on the scale and feedback. After that, an average score is calculated for each LLM configuration based on the interview scores.

4. As a final step, the results were compared and analyzed. From a qualitative perspective, the average scores obtained from the interviews serve as an indicator of which LLM is able to produce meaningful results with minimal input during the threat validation process. These scores also indicate which LLM the participants believe provides the most relevant and comprehensive threat descriptions.

   The quantitative analysis completes the qualitative results by analyzing whether the observed qualitative benefits align with the performance metrics. In particular, the

computing power and time efficiency of the various LLMs are evaluated to ensure that the most effective qualitative solutions are also good solutions in terms of performance.

With this approach, the evaluation aims to assess the efficiency of the LLM-based threat modeling approach for both threat identification and validation while identifying the most effective configurations for a RAG system with minimal input.

5.2 Analysis

This section focuses on the results of the quantitative and qualitative analysis. The results are presented, and the most important elements of the findings are discussed. The section is divided into two parts: Threat identification, in which the quantitative and qualitative results of the threat identification evaluation are examined, and threat validation, in which the qualitative and quantitative results of the threat validation are analyzed.

5.2.1 Threat Identification

To evaluate whether the prototype can identify threats with minimal input, predefined scenarios with a given DFD are used. This DFD was manually analyzed by the interview participants to identify potential threats. The resulting threat model is considered the optimal reference model. Based on this, the prototype is evaluated to determine which configurations perform best in identifying threats with minimal inputs.

The scenario and the related DFD can be found in Appendix C. To briefly characterize the used scenarios, it is worth noting that the first scenario concerned an AI application that analyzes patient data to predict future health problems. In this scenario, the participants considered the role of a patient in the application, which is why confidentiality was of utmost importance. The second scenario involved a banking application that uses an external AI application to detect anomalies in its application. In this scenario, the participants took on the role of the bank manager, with integrity being the main concern as they want the integration of the AI application to work as intended. The threats mentioned by the interviewees are listed in Table 5.1, which lists the four most frequently mentioned threats for each scenario.

- R1.1: Model Inversion and Membership Inference (4 votes)
- R1.2: Leak sensitive input data (3 votes)
- R1.3: Sensitive data output from model (2 votes)
- R1.4: Development-time data leak (2 votes)
- R2.1: Data Poisoning (3 votes)

- R2.2: Open-box evasion (3 votes)
- R2.3: Supply-chain model poisoning (3 votes)
- R2.4: Evasion after data poisoning (3 votes)

The analysis of the identified threats shows clear patterns between Scenario 1 and Scenario 2. In Scenario 1, one more threat was selected and there was less alignment among the respondents than in Scenario 2. A total of nine threats were identified in Scenario 1. All four respondents recognized *Model Inversion and Membership Inference*, followed by *Leak sensitive input data*, which was selected by three respondents. In addition, *Sensitive data output from the model* and *Development-time data leak* were each mentioned by two respondents. The remaining five threats were only mentioned once and are therefore excluded from the further evaluation.

	Threat	Respondent 1	Respondent 2	Respondent 3	Respondent 4
Scenario 1	**Threat 1**	Sensitive Data Output from Model	Development-time data leak	Runtime model poisoning	Model inversion and Membership inference
	Threat 2	Data Poisoning	Leak sensitive input data	Sensitive data output from model	Development-time data leak
	Threat 3	Leak sensitive input data	Evasion after data poisoning	Model inversion and Membership inference	Source code / configuration leak
	Threat 4	Model Inversion and Membership Inference	Model inversion and Membership inference	Model theft through model parameter leak	Leak sensitive input data
Scenario 2	**Threat 1**	Supply-chain Model Poisoning	Data poisoning	Open-box evasion	Open-box evasion
	Threat 2	Insecure Output Handling	Evasion after data poisoning	Evasion after data poisoning	Evasion after data poisoning
	Threat 3	Runtime Model Poisoning	Open-box evasion	Supply-chain model poisoning	Data poisoning
	Threat 4	Data Poisoning	Model theft through model parameter leak	Model theft through use	Supply-chain Model Poisoning

Table 5.1: Overview of Threat Relevance by Scenario and Interviewees

Eight threats were identified in Scenario 2. In contrast to scenario 1, four threats were highlighted by three respondents, which indicates a higher level of consensus among the participants. The remaining four threats were only mentioned by individual respondents and are therefore also not included in the evaluation. A detailed list of which respondents chose which threats can be found in Appendix D.

Table 5.2 and Table 5.3 present the qualitative and quantitative results of the threat identification evaluation. The first column in the tables defines the similarity search strategy, followed by the second column, which lists the chunking strategies described in Section 5.1.3. In the third column, the number of threats that need to be identified k is variable in order to observe whether a higher k leads to better results. The column *# threat* indicates the total number of threats identified for the scenario in relation to the

	Chunking Strategy	k	R1.1	R1.2	R1.3	R1.4	# Threats	TPR	∅ Time
Low-depth k	**Default**	4	yes	yes	yes	no	3	100%	3.07s
		6	yes	yes	yes	no	5	60%	3.02s
	Manual Pre-Processing	4	yes	no	yes	yes	5	60%	2.72s
		6	yes	no	yes	yes	5	60%	2.80s
	Recursive Chunking	20	no	yes	yes	yes	3	100%	2.85s
		25	yes	yes	yes	yes	5	80%	2.80s
	Summarization	4	yes	yes	no	no	4	50%	2.83s
		6	yes	yes	no	no	4	50%	2.69s
High-depth k	**Default**	7	yes	yes	yes	no	5	60%	1.78s
		8	yes	yes	yes	yes	6	66.7%	1.80s
	Manual Pre-Processing	7	yes	yes	yes	no	5	60%	1.52s
		8	yes	yes	yes	no	5	60%	1.53s
	Recursive Chunking	50	yes	yes	yes	no	6	50%	1.48s
		60	yes	yes	yes	yes	7	57%	1.48s
	Summarization	8	yes	no	yes	no	3	66.7%	1.45s
		15	yes	yes	yes	no	6	50%	1.39s
Reranking		25	yes	yes	yes	yes	5	80%	N/A

Table 5.2: Identification of Threats by Prototype Configuration for Scenario 1

respective CIA triad property. In the interviews, the interviewees were given a description of each scenario and the necessary perspective to identify threats. In contrast, the tool does not rely on text input but works only on the basis of the DFD, while the frontend allows filtering according to the elements of the CIA triad to prioritize different security aspects. Scenario 1 prioritizes confidentiality, as respondents took the perspective of patients, while scenario 2 prioritizes integrity, reflecting the perspective of an e-banking manager, as further explained in Appendix D. The *Rate* column shows the ratio between the threats considered important by the respondents and the threats identified by the tool. The ∅ *Time* column shows the average time taken by the tool to identify threats in at least five attempts.

By analyzing both tables simultaneously, it becomes clear that the recursive chunking strategy delivered the best qualitative results when comparing the rates. In Scenario 1, however, the default chunking strategy delivered competitive results, while this was not the case in Scenario 2. Recursive chunking is the only chunking strategy that recognized a similar number of threats in both scenarios and identified all four threats mentioned by the respondents. However, in Scenario 1, the *low-depth* k similarity search strategy performed better, while in Scenario 2, the *high-depth k* strategy was dominant. Furthermore, the True Positive Rate (TPR) in Scenario 2 is generally lower than in Scenario 1. The reason for this difference is that for certain integrity-related threats, several threats are displayed in the

	Chunking Strategy	k	R2.1	R2.2	R2.3	R2.4	# Threats	TPR	∅ Time
Low-depth k	Default	4	no	no	no	no	3	0%	3.35s
		12	no	yes	no	no	6	16.7%	3.43s
	Manual Pre-Processing	4	no	yes	no	yes	9	22.2%	2.94s
		6	yes	yes	yes	yes	12	33.3%	2.80s
	Recursive Chunking	15	yes	yes	no	yes	7	42.9%	2.90s
		25	yes	yes	yes	yes	9	44.4%	3.1s
	Summarization	4	yes	yes	no	yes	7	42.9%	2.82s
		6	yes	yes	no	yes	7	42.9%	2.78s
High-depth k	Default	7	no	yes	no	no	4	25%	1.98
		15	no	yes	no	no	6	16.7%	1.80s
	Manual Pre-Processing	7	no	no	yes	no	1	100%	1.65s
		15	yes	no	yes	yes	6	50%	1.65s
	Recursive Chunking	20	yes	yes	no	yes	5	60%	1.67s
		30	yes	yes	yes	yes	7	57.1%	1.71s
	Summarization	8	no	yes	yes	yes	9	33.3%	1.62s
		10	no	yes	yes	yes	10	30%	1.53s
Reranking		200	yes	yes	yes	yes	10	40%	N/A

Table 5.3: Identification of Threats by Prototype Configuration for Scenario 2

frontend even though only one was identified by the backend. For example, if the threat *Prompt Injection* is identified, both *Direct Prompt Injection* and *Indirect Prompt Injection* appear in the frontend, which lowers the rate, especially if *Prompt Injection* itself was not explicitly mentioned. This design decision and its implementation in ThreatFinderAI are explained in detail in the Section 4.3. Consequently, the goal was to achieve a number of around five threats in Scenario 1 and between six and seven in Scenario 2. The recursive chunking strategy successfully achieved this goal by identifying five threats in Scenario 1, covering all four threats considered relevant by the interviewees, and seven threats in Scenario 2, identifying all four interview-relevant threats.

In Scenario 1, the similarity search with *low-depth k* and recursive chunking were used for reranking to see if the results could be improved. A k value of 25 was chosen, and the 110 best results were used as output. It is important to note that the same threats appeared many times in the output, and therefore, the number is so high. As described in Section 4.4.5, this is due to the fact that the threats are stored in the metadata and not in the description. However, the quality of the output in Scenario 1 remains the same with and without reranking.

Scenario 2 becomes relevant as the reranking was performed with the *high-depth k* similarity search in combination with the recursive chunking strategy, and it does not have the

same TPR as in Scenario 1. In this case, k was set to 200, but only the top 10 threats were considered, as they are not identical in the *high-depth k* strategy. Only when the top 10 results were selected, all threats could be identified, which was a criterion for comparing this approach with recursive chunking without reranking. However, the detection TPR with reranking was only 40%, while it was 57.1% without reranking. It is worth mentioning that the *Prompt Injection* and *Evasion* threats were identified in Scenario 2. This indicates that two additional threats were identified due to the frontend rule explained in Section 4.3. It could be argued that the actual detection TPR is closer to 50%. However, as this adjustment was not taken into account in the other values as well, it must also be treated consistently in the reranking evaluation.

From a quantitative point of view, it is clear that the average time required for threat identification is significantly lower compared to threat validation. In addition, the *high-depth k* similarity search strategy generally works twice as fast as the *low-depth k* strategy. This difference is due to the fact that the *low-depth k* strategy has to go through all objects and determine k threats for each object, which naturally increases the processing time.

5.2.2 Threat Validation

The threat validation evaluation determines whether the prototype can validate threats with minimal input. Specifically, it evaluates whether and how effectively an LLM-based approach can identify a threat at a specific point in the system and describe how the threat could be exploited in this scenario. It also examines which LLM best fulfills this task in terms of quality and performance. The LLMs evaluated in this thesis are shown in Table 5.4 together with the Likert score given by the respondents. A score of 1 indicates poor validation. For example, the response contains no important information, while a score of 5 indicates very effective validation, providing all the necessary details, such as the description of the threat, its location, and how it could be exploited in the given scenario. In addition, Appendix D provides insight into the reasons for the scores given by respondents, highlighting aspects that they found particularly useful and areas where validation was insufficient.

LLM	Respondent 1	Respondent 2	Respondent 3	Respondent 4	∅
Llama 3.2 (3B)	3.5	3	3.5	3.5	3.375
Deepseek-R1 (1.5B)	2.5	2	2	2	2.125
Deepseek-R1 (7B)	3.5	4	3	3	3.375
Mistral (7B)	3.5	3.5	3	3.5	3.375
Llama 3.1 (8B)	2.5	2.5	2	2.5	2.375
Gemini 1.5 Flash	4.5	5	3	4	4.125

Table 5.4: Average Interviewee Likert Scale for Threat Description Accuracy by LLMs

Table 5.4 shows that the LLM with the highest average Likert score in both scenarios is Gemini 1.5 Flash. The interviewees especially appreciated the contextualized descriptions

that included terminologies from the DFD and specific references to the architectural building blocks. Llama 3.2, Mistral, and DeepSeek-R1 (7B) achieved the second-highest average rating. Llama 3.2 was appreciated for its clear and simple explanations. However, in Scenario 1, the description was considered too extensive and difficult to read. This was because Llama 3.2 first explained the danger of *model inversion* and then described the *membership inference* separately, which is illustrated in Appendix D. DeepSeek-R1 (7B) was also praised for its concise descriptions that maintained a link to the DFD, but interviewees noted that it sometimes lacked a concrete explanation of the impact of the exploit. In contrast, Mistral provided short answers and clearly described the impact of the exploitation. However, the respondents noted that Mistral did not specify exactly where the system is targeted with the specific threat. Llama 3.1 received the second-lowest score among all LLMs. The main problem was that it referred to the trust boundary without further explaining it, and simply stated that the trust boundary was compromised. This led to confusion and misleading information for the respondents, as the trust boundary itself was not the real issue. DeepSeek-R1 (1.5B) received the lowest score in this evaluation. The reason for this was the generation of false information describing a threat that was not relevant to the given scenario. As a result, this LLM often received a score of 1 on the Likert scale, which explains its low average score.

LLM	∅ Response Time	∅ Error Rate	GPU utilization
Llama 3.2 (3B)	28.2s	0%	85%
Deepseek-R1 (1.5B)	18.7s	20%	75.6%
Deepseek-R1 (7B)	103.0s	60%	48%
Mistral (7B)	40.8s	0%	93%
Llama 3.1 (8B)	70.0s	100%	35.8%
Gemini 1.5 Flash	5.1s	0%	N/A

Table 5.5: Quantitative Evaluation of LLM Performance in Threat Validation

In addition to qualitative performance, Gemini 1.5 Flash also showed the best efficiency, as shown in Table 5.5. It achieved the fastest average response time of 5.1 seconds and had an error rate of 0%. This is likely due to the fact that Gemini 1.5 Flash was the only LLM that was not self-hosted but accessed via an API. The other LLMs were self-hosted on a GeForce GTX 1660 SUPER graphics card, which means that response times and GPU utilization should be taken with a grain of salt, as users deploying the prototype in practice would likely have access to more powerful consumer-grade hardware. However, the trend is clear: smaller LLMs generally have lower response times. Furthermore, only the DeepSeek models and Llama 3.1 had an error rate of more than 0%. This means that their output does not comply with the formatting which is required for the prototype. In the case of DeepSeek, this error rate could be reduced quickly, as the outputs still contain unnecessary tags such as <think>, which the prototype was unable to process. Simply filtering out these parts of the response could improve the error rating. There was also an issue with Llama 3.1, where the model kept adding explanation sentences at the

beginning and end of its response, even though the prompt clearly stated the expected output format. The prototype was unable to process these additional explanations. This problem initially occurred with Llama 3.2 during the development of the prototype but was resolved by customizing the prompt. This same prompt structure seemed to work effectively for Llama 3.2, Mistral, and Gemini 1.5 Flash.

In terms of GPU utilization, LLMs such as Mistral and Llama 3.2, were operating at the GPU upper capacity limit, while others, such as Llama 3.1 and DeepSeek-R1, had extra capacity. This suggests that the latter models could theoretically run on weaker GPU hardware and achieve similar response times. However, models such as Mistral and Llama 3.2 would probably have longer response times with less powerful hardware.

5.3 Discussion

This section discusses the findings of the evaluation and is divided into three subsections: Threat Identification, Threat Validation, and Conclusion. Possible reasons for the effectiveness of the identification strategies are explained, the influence of minimal input on reranking is discussed, and the trade-offs between cloud-based and self-hosted LLMs are highlighted. Finally, the main findings are summarized, and possible improvements are suggested.

5.3.1 Threat Identification

From the results of the interviews, it can be stated that the recursive chunking strategy performs best in terms of quality when identifying threats with minimal input. The main reason for this is that the critical threats that resulted from the interviews were identified effectively in the scenarios, as shown by the identification TPR in Tables 5.2 and Table 5.3. However, these results should be interpreted with caution, as only the four threats most frequently mentioned by respondents were taken into account, while all others were excluded. This exclusion may have led to an incomplete picture of the overall threat environment.

However, considering the evaluation method used, it can be seen that the recursive chunking strategy in combination with the *low-depth k* and *high-depth k* similarity search works well in identifying relevant threats despite minimal input. This result is surprising, as the original hypothesis was that threat identification would improve with manual preprocessing or summarization to ensure a consistent basis for all threats. One possible explanation is that only a small part of a threat description is actually relevant for identification. Since recursive chunking splits the threat description into several smaller parts, these key segments can be better captured. On the other hand, if embeddings are generated from the full description, these relevant sections can be diluted.

Initially, it was assumed that manual pre-processing would solve this problem by retaining only the information that directly describes the threat. However, it is possible that additional contextual information, such as security controls, may have played an important

role in identification. Furthermore, it is also possible that even after pre-processing, the remaining descriptions were too long, making it difficult for the retrieval step to detect the details that are important for identification.

As for the summarization, one explanation for the lower-than-expected performance could be that the LLM in this prototype had limited information about each threat, unlike other general RAG systems. In some cases, the LLM may have had to generate additional information to fulfill the token constraints, which defeats the intended purpose of this strategy. This could be an explanation for the relatively low performance. In contrast, RAG systems typically rely on large KBs, making summarization a more feasible approach. However, in this prototype, the threat information came from a single source, so the available knowledge was already limited.

Another surprising finding was that the reranking approach did not improve the results. An important observation in the reranking approach was the consistently low relevance score by the LLM. The highest recorded score was only 0.257 (on a scale of 0 to 1), which is relatively low. Two possible explanations come to mind: Either the query wording was insufficient, or the embeddings used in the similarity search were not optimized. As this study focuses on identifying threats with minimal inputs, it is likely that the query itself was poorly structured as it relied only on the DFD. This limitation suggests that a reranking approach would have potential if more contextual information were available. One example would be to include the CIA attributes in the query, which could improve the results. In the current version of the prototype such filters are not integrated in the backend but in the frontend. Alternatively, a brief system description or scenario context could be optionally included to increase the relevance score, similar to what the respondents had in the interview. With this additional context, the reranking model could assess threats more effectively. Although the reranking approach shows potential, it does not provide any added value when applied to minimal inputs, such as a single DFD, as demonstrated in this thesis.

Performance Considerations Looking at the average duration of threat detection, it is noteworthy that the *high-depth k* similarity search strategy is faster than the *low-depth k* approach. The reason for this is that for the *low-depth k* strategy, the similarity search is done individually for each asset. Small DFDs with 5-20 assets result in a delay of only 1-4 seconds compared to the high-depth strategy. However, in large DFDs which can occur in practical settings, the time difference can become larger, making the *high-depth k* strategy more efficient. This is because the *high-depth k* strategy performs a single similarity search, avoiding repeated computations.

For the chunking strategies, the execution time did not vary significantly, nor did using different k values have a significant effect on performance. Therefore, the similarity search strategy itself is the most important factor affecting efficiency.

5.3.2 Threat Validation

For threat validation, the qualitative and quantitative results of a cloud-based solution with Gemini 1.5 Flash are the most suitable. However, it is important to consider that

cloud-based solutions have security implications and each organization must decide for itself whether to go to the cloud. In a cloud, sensitive data such as a DFD is no longer stored locally, which can lead to security risks. For organizations that already store their data and DFDs in the cloud, a cloud-based LLM is a logical choice due to its cost efficiency. Gemini 1.5 Flash, for example, costs $0.0000375 per 1000 characters per request, which is significantly cheaper than running an on-premise LLM when it is not regularly used [70].

However, for companies that do not use cloud storage for sensitive data, running an LLM onsite may be a better alternative. In this case, Llama 3.2 or Mistral would be the recommended models for this prototype. Both LLMs delivered comparable qualitative results and outperformed other self-hosted options in terms of performance. In particular, both models achieved an error rate of 0%, with Llama 3.2 having an average response time of 28 seconds and Mistral of 41 seconds. Given the hardware limitations and the complexity of the task, these response times are acceptable. In terms of performance, Llama 3.2 seems to be the better choice. However, Mistral, with its 7 billion parameters, has more potential for larger and more complex use cases and could most likely do much more in terms of activities. It is important to note that this prototype was originally developed with Llama 3.2 as the default model, which means that prompts and system configurations have been optimized for this model. This means that adapting the prompts for Mistral or DeepSeek-R1 (7B) could provide even better results and possibly outperform Llama 3.2. In addition, for DeepSeek-R1 (7B), further improvements in output processing could reduce the error rate, improve overall performance, and potentially close the gap in quality performance compared to Mistral.

5.3.3 Conclusion

To conclude, threat identification using a recursive chunking strategy in combination with a *high-depth k* similarity search works well even with minimal input. However, for an input that is limited to a DFD, the reranking approach does not lead to better identification results. Nevertheless, it has significant potential when enhanced with a small amount of additional data.

For threat validation, organizations must decide whether to use a cloud-based LLM or a self-hosted solution. While Gemini 1.5 Flash delivers better quality and more powerful results, it has the disadvantage of limited data control. If an organization prefers to host its own LLM without modifying the prototype, Llama 3.2 is currently the best option in terms of quality and performance, assuming the same infrastructure is used. However, if minor modifications to the prototype are acceptable, Mistral and DeepSeek-R1 (7B) show the most potential.

The evaluation results indicate that Mistral excels at describing the detailed impact of a threat, while DeepSeek-R1 (7B) is particularly effective at providing concise threat descriptions and accurately identifying potential attack vectors. While Mistral was weaker in the latter area, it remains a strong candidate for organizations that require detailed threat analysis.

In a broader context, these results demonstrate a use case for AI in cybersecurity. The prototype demonstrates how minimal inputs can be used to effectively identify and validate specific AI threats, making it relevant for organizations of all sizes, from small to large. Furthermore, this work shows that even smaller open-source models such as Mistral 7B and DeepSeek-R1 (7B) can deliver strong results, highlighting the importance and competence of open-source AI. This is particularly important in terms of data security and privacy, as open source solutions provide a customizable and secure alternative, as organizations retain full control over the model, its data, and its deployment. This study not only demonstrates that threat identification and validation is possible, but also encourages further innovation in cybersecurity by exploring how AI can support a broader range of cybersecurity tasks.

Chapter 6

Summary, Conclusions and Future Work

This thesis investigated the question whether an LLM approach can effectively support the threat modeling to AI systems. To this end, a comprehensive literature review was conducted in which existing research on this topic was analyzed. It was found that current approaches mainly rely on extensive documentation and intermediary structures, which are not suitable for early-stage threat modeling due to their elaborated nature. Furthermore, there are also existing studies focusing on the use of LLMs for threat validation and not on threat identification, which is also an important aspect of threat modeling. As a result, a major limitation was identified: There is no existing tool that attempts to perform threat identification and threat validation with minimal inputs for AI system threats.

To close this gap, an approach was designed and prototypically developed that performs both threat identification and validation using a RAG system. The approach follows a two-step process that allows these tasks to be evaluated separately. In addition, threat validation is particularly error-prone due to the results generated by the LLM and would need to be further improved for real-world applications. However, the primary goal of this work was not to develop a production-ready tool, but rather to investigate the feasibility of threat identification and validation with minimal input.

The prototype builds on an existing work [3] that statically identifies threats based on assets. This existing tool serves as a frontend, while a newly developed backend interacts with a vector database and LLMs to enhance the identification and validation capabilities.

Both threat identification and validation were evaluated independently. A qualitative evaluation was conducted through interviews with four different users to determine the most effective RAG configuration and LLMs for this task. In addition, the performance of the different configurations and LLMs was measured.

The results of the evaluation show that threat identification using recursive chunking with a *high-depth k* similarity search works well even with minimal inputs. The evaluation showed that this approach was able to identify important threats in fictional scenarios. In addition, threat validation also performed well according to feedback from respondents. The participants noted that the explanations provided by the prototype were relevant

to the AI threats and the fictional DFD. This concludes that the use of AI in threat modeling holds potential and is able to process tasks traditionally performed by experts. Consequently, AI can serve as a supporting tool for threat modeling aimed at different stakeholders. For example, it can help software architects who do not have security expertise but are involved in software planning, as well as security engineers who can use the AI tool to look at the threats from alternative perspectives.

6.1 Future Work

This section shows what future research topics could be based on this work. The future work can be divided into three areas:

Prototype Improvements

The current prototype is primarily designed for research and testing purposes and not for production use. It does not yet support large-scale retrievals or strict performance requirements. In order to minimize dependencies and ensure a quick proof of concept, simpler implementation approaches were chosen. However, efficiency improvements are essential to turn the prototype into a production-ready system. One solution is the integration of LangChain Expression Language (LCEL), which is a declarative method for building chains ranging from simple combinations to complex multi-step workflows [71]. Implementing LCEL could significantly improve the performance of the system in a production environment, as asynchronous execution and optimized parallel processing are a major focus of this language.

Another crucial improvement is the implementation of a better-structured output format from the LLM. Currently, the formatting of the output is given by prompts, but this method is not always reliable and makes further processing difficult. This is a known issue, and LangChain has introduced a structured output approach where the models directly generate responses in a predefined format [72]. The implementation of this approach is important to make the prototype more robust and reliable beyond experimental use.

Another improvement would be the extension of further KB. The current version of the prototype only supports Markdown files from [53] to embeddings, which would need to be changed for such an enhancement. This would require a change to the chunker module of the backend to support other data formats such as PDFs.

In addition, the current prototype identifies and evaluates all threats relevant to a specific DFD. The frontend allows users to prioritize certain properties of the CIA triad and filter the threats accordingly. However, passing this prioritization information to the backend would provide more context to the LLM for threat identification and validation. Future work could investigate how well an LLM can utilize this additional information, or if a predefined mapping is necessary to determine which threats are most critical for each CIA property.

Evaluation Methods

Since AI security threat identification and validation using RAG with minimal inputs is not well researched, there are many unexplored opportunities for experiments. This work mainly focused on two methods of similarity search, chunking strategies, reranking techniques, and different self-hosted LLMs. However, future research could also investigate other methods, such as:

- Alternative embedding models
- Different vector databases
- Different similarity search algorithms
- Improved prompt engineering methods
- Integration with external data sources such as BigQuery

Another important area for future evaluation is hardware performance. An organization using a self-hosted LLM would need access to much more powerful hardware than was available for this work. Testing with such infrastructure would enable the use of LLMs with larger parameters, allowing them to compete with cloud-based models in terms of quality and efficiency.

Furthermore, the ranking system used in threat validation was not evaluated in this thesis. The prototype assigns a ranking to each identified threat during the validation process. The ranking system can influence the evaluation results of different self-hosted LLMs, as some models could handle the ranking qualitatively better than others. This capability should be introduced as an additional qualitative metric to further define and optimize the selection of LLMs.

In addition, it would be useful to evaluate the performance of the prototype on large DFDs with multiple assets. The current evaluation focused on smaller DFDs, but it remains uncertain whether the quality and performance of the prototype can be scaled to larger, more complex architectures.

Alternative Approaches

There are numerous directions for future work beyond those described in this section. The ideas proposed are still based on fundamental assumptions, such as the use of a RAG-based approach. However, alternative strategies could also be explored. For example, if large training data were available, fine-tuning an LLM specifically for AI threat modeling could be a feasible option. This approach would only be feasible for organizations with access to large datasets, as publicly available AI threat models are rare.

Alternatively, another approach could be to put all the knowledge about AI security threats into a single prompt and compare the model's responses with those generated using a RAG system. As the prototype currently processes a relatively small dataset, it is theoretically possible to put all the relevant knowledge into a single prompt, especially

if LLMs continue to extend their context length. This experiment would show how an LLM, in contrast to a RAG system, can identify and validate threats with minimal input. However, this method would always be limited by the context length restriction. As system descriptions become more detailed and KBs grow, this approach may eventually reach its limits. Nevertheless, it would be interesting to investigate whether this method delivers comparable, better, or worse results than the RAG-based method.

Finally, AI-assisted threat modeling remains an evolving field, and this work serves as a foundation for further research. Future work should aim to further improve and extend these ideas to explore the boundaries of what is possible in AI security with the support of LLMs.

Bibliography

[1] "OWASP software assurance maturity model". (Sep. 13, 2023), [Online]. Available: https://owasp.org/www-project-samm/ (visited on 03/07/2025).

[2] A. Kucharavy, Z. Schillaci, L. Maréchal, *et al.*, *Fundamentals of generative large language models and perspectives in cyber-defense*, Mar. 21, 2023.

[3] J. von der Assen, J. Sharif, C. Feng, C. Killer, G. Bovet, and B. Stiller, "Asset-centric threat modeling for ai-based systems", in *2024 IEEE International Conference on Cyber Security and Resilience (CSR)*, 2024, pp. 437–444.

[4] S. Hussain, A. Kamal, S. Ahmad, G. Rasool, and S. Iqbal, "Threat modelling methodologies: A survey", *Sci. Int.(Lahore)*, vol. 26, no. 4, pp. 1607–1609, 2014.

[5] N. Shevchenko, T. A. Chick, P. O'Riordan, T. P. Scanlon, and C. Woody, "Threat modeling: A summary of available methods", *Software Engineering Institute| Carnegie Mellon University*, pp. 1–24, 2018.

[6] D. S. Cruzes, M. G. Jaatun, K. Bernsmed, and I. A. Tøndel, "Challenges and experiences with applying microsoft threat modeling in agile development projects", in *2018 25th Australasian Software Engineering Conference (ASWEC)*, IEEE, 2018, pp. 111–120.

[7] Z. Braiterman, A. Shostack, J. Marcil, *et al.* "Threat modeling manifesto". (Nov. 17, 2020), [Online]. Available: https://www.threatmodelingmanifesto.org/ (visited on 03/07/2025).

[8] V. Drake. "Threat modeling | OWASP foundation", OWASP. (Mar. 31, 2024), [Online]. Available: https://owasp.org/www-community/Threat_Modeling (visited on 03/07/2025).

[9] I. Tarandach and M. Coles, *Threat Modeling: A Practical Guide for Development Teams*. O'Reilly Media, Dec. 22, 2020, 245 pp.

[10] "MITRE ATLAS". (Jun. 2021), [Online]. Available: https://atlas.mitre.org/ (visited on 03/07/2025).

[11] A. Marshall, R. Rojas, J. Stokes, and D. Brinkman. "Securing the future of AI and ML at microsoft". (Jan. 22, 2024), [Online]. Available: https://learn.microsoft.com/en-us/security/engineering/securing-artificial-intelligence-machine-learning (visited on 03/07/2025).

[12] R. van der Veer. "OWASP AI security and privacy guide | OWASP foundation". (Mar. 2025), [Online]. Available: https://owasp.org/www-project-ai-security-and-privacy-guide/ (visited on 03/07/2025).

[13] "Securing machine learning algorithms | ENISA". (Feb. 21, 2024), [Online]. Available: https://www.enisa.europa.eu/publications/securing-machine-learning-algorithms (visited on 03/07/2025).

[14] "What is a large language model (LLM)?", Cloudflare. (Sep. 30, 2024), [Online]. Available: https://www.cloudflare.com/learning/ai/what-is-large-language-model/ (visited on 03/07/2025).

[15] "What are large language models (LLMs)? | IBM". (Nov. 2, 2023), [Online]. Available: https://www.ibm.com/topics/large-language-models (visited on 03/07/2025).

[16] D. Bergmann and C. Stryker. "What is an attention mechanism? | IBM". (Dec. 5, 2024), [Online]. Available: https://www.ibm.com/think/topics/attention-mechanism (visited on 03/07/2025).

[17] J. Kaddour, J. Harris, M. Mozes, H. Bradley, R. Raileanu, and R. McHardy, *Challenges and applications of large language models*, Jul. 2023.

[18] "Retrieval augmented generation (RAG) | LangChain". (Nov. 26, 2024), [Online]. Available: https://python.langchain.com/docs/concepts/rag/ (visited on 03/07/2025).

[19] I. Belcic and C. Stryker. "RAG vs. fine-tuning | IBM". (Aug. 14, 2024), [Online]. Available: https://www.ibm.com/think/topics/rag-vs-fine-tuning (visited on 03/07/2025).

[20] Belcic. "What is RAG (retrieval augmented generation)? | IBM". (Oct. 21, 2024), [Online]. Available: https://www.ibm.com/think/topics/retrieval-augmented-generation (visited on 03/07/2025).

[21] A. Gutowska. "Chunking strategies for RAG tutorial using granite | IBM". (Jan. 22, 2025), [Online]. Available: https://www.ibm.com/think/tutorials/chunking-strategies-for-rag-with-langchain-watsonx-ai (visited on 03/07/2025).

[22] F. Pedrazzini. "Chunking strategies in retrieval-augmented generation (RAG) systems", Prem. (Sep. 17, 2024), [Online]. Available: https://blog.premai.io/chunking-strategies-in-retrieval-augmented-generation-rag-systems/ (visited on 03/07/2025).

[23] "Rerankers and two-stage retrieval | pinecone". (Oct. 18, 2023), [Online]. Available: https://www.pinecone.io/learn/series/rag/rerankers/ (visited on 03/07/2025).

[24] "OpenAI platform". (), [Online]. Available: https://platform.openai.com (visited on 03/07/2025).

[25] "Accuracy vs. precision vs. recall in machine learning: What is the difference?" (Nov. 23, 2023), [Online]. Available: https://encord.com/blog/classification-metrics-accuracy-precision-recall/ (visited on 03/07/2025).

[26] L. Monigatti. "Evaluation metrics for search and recommendation systems | weaviate". (May 28, 2024), [Online]. Available: https://weaviate.io/blog/retrieval-evaluation-metrics (visited on 03/07/2025).

[27] "Classification: Accuracy, recall, precision, and related metrics | machine learning", Google for Developers. (Mar. 3, 2025), [Online]. Available: `https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall` (visited on 03/07/2025).

[28] N. F. Liu, K. Lin, J. Hewitt, *et al.*, "Lost in the middle: How language models use long contexts", *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024.

[29] "Enhancing RAG models with reranking & LangChain". (May 22, 2024), [Online]. Available: `https://myscale.com/blog/maximizing-advanced-rag-models-langchain-reranking-techniques/` (visited on 03/07/2025).

[30] D. Andrés and J. Ferrer. "Issue #79 - optimize RAG with reranking", Machine Learning Pills. (Nov. 3, 2024), [Online]. Available: `https://mlpills.substack.com/p/issue-79-optimize-rag-with-reranking` (visited on 03/07/2025).

[31] J. Shin. "Cross encoder reranker | LangChain OpenTutorial". (Jan. 21, 2025), [Online]. Available: `https://langchain-opentutorial.gitbook.io/langchain-opentutorial/11-reranker/01-crossencoderreranker` (visited on 03/07/2025).

[32] T. Mishra, E. Sutanto, R. Rossanti, *et al.*, "Use of large language models as artificial intelligence tools in academic research and publishing among global clinical researchers", *Scientific Reports*, vol. 14, no. 1, p. 31 672, 2024.

[33] I. Elsharef, "Large language model assisted threat modeling", M.S. thesis, The University of Wisconsin-Milwaukee, 2023.

[34] S. Rajapaksha, R. Rani, and E. Karafili, "A rag-based question-answering solution for cyber-attack investigation and attribution", *arXiv preprint arXiv:2408.06272*, 2024.

[35] M. Hassanin, M. Keshk, S. Salim, M. Alsubaie, and D. Sharma, "Pllm-cs: Pre-trained large language model (llm) for cyber threat detection in satellite networks", *Ad Hoc Networks*, vol. 166, p. 103 645, 2025.

[36] Y. Sun, D. Wu, Y. Xue, *et al.*, "Llm4vuln: A unified evaluation framework for decoupling and enhancing llms' vulnerability reasoning", *arXiv preprint arXiv:2401.16185*, 2024.

[37] S. Majumdar and T. Vogelsang, "Towards safe llms integration", *Large*, p. 243, 2024.

[38] O. Gadyatskaya and D. Papuc, "Chatgpt knows your attacks: Synthesizing attack trees using llms", in *International Conference on Data Science and Artificial Intelligence*, Springer, 2023, pp. 245–260.

[39] V. Tanksale, "Cyber threat hunting using large language models", in *International Congress on Information and Communication Technology*, Springer, 2024, pp. 629–641.

[40] M. Hassanin and N. Moustafa, "A comprehensive overview of large language models (llms) for cyber defences: Opportunities and directions", *arXiv preprint arXiv:2405.14487*, 2024.

[41] M. A. Ferrag, F. Alwahedi, A. Battah, B. Cherif, A. Mechri, and N. Tihanyi, "Generative ai and large language models for cyber security: All insights you need", *Available at SSRN 4853709*, 2024.

[42] A. Ghosh, "AI-enhanced cyber security: Leveraging large language models for threat detection", *Innovative Computer Sciences Journal*, vol. 9, no. 1, Oct. 14, 2023, Number: 1, ISSN: 3007-6471. [Online]. Available: https://innovatesci-publishers.com/index.php/ICSJ/article/view/211 (visited on 10/12/2024).

[43] Y. Liu, S. Li, X. Wang, and L. Xu, "A review of hybrid cyber threats modelling and detection using artificial intelligence in iiot", *Computer Modeling in Engineering & Sciences*, vol. 140, no. 2, 2024.

[44] D. R. Chittibala, "Threat model detection using ai", *INTERNATIONAL JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH AND DEVELOPMENT (IJAIRD)*, vol. 2, no. 1, pp. 40–47, 2024.

[45] I. Elsharef, Z. Zeng, and Z. Gu, "Facilitating threat modeling by leveraging large language models", in *Workshop on AI Systems with Confidential Computing*, 2024.

[46] W. B. Mbaka and K. Tuma, "Usefulness of data flow diagrams and large language models for security threat validation: A registered report", *arXiv preprint arXiv:2408.07537*, 2024.

[47] A. Chiş, O. I. Stoica, A.-M. Ghiran, and R. A. Buchmann, "A knowledge graph approach to cyber threat mitigation derived from data flow diagrams", in *2024 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, 2024, pp. 1–6.

[48] Y. Chen, M. Cui, D. Wang, *et al.*, "A survey of large language models for cyber threat detection", *Computers & Security*, p. 104 016, 2024.

[49] AppSecEngineer, *Webinar: Rapid threat modeling with GenAI and LLMs*, Apr. 11, 2024. [Online]. Available: https://www.youtube.com/watch?v=ZNWptwfa0DE (visited on 03/07/2025).

[50] S. Diemert and J. H. Weber, "Can large language models assist in hazard analysis?", in *International Conference on Computer Safety, Reliability, and Security*, Springer, 2023, pp. 410–422.

[51] "Leveraging LLMs for threat modeling - GPT-3.5 vs claude 2 vs GPT-4", xvnpw personal blog. Section: posts. (Sep. 3, 2023), [Online]. Available: https://xvnpw.github.io/posts/leveraging-llms-for-threat-modelling-gpt-3.5-vs-claude2-vs-gpt-4/ (visited on 03/07/2025).

[52] M. AboElKheir. "I asked "ChatGPT" how to threat model features using LLM", AppSec Untangled. (Jul. 1, 2023), [Online]. Available: https://medium.com/appsec-untangled/i-asked-chatgpt-how-to-threat-model-features-using-llm-8477600445d (visited on 03/07/2025).

[53] "AI exchange". (2025), [Online]. Available: https://owaspai.org/ (visited on 03/07/2025).

[54] "LangChain". (2022), [Online]. Available: https://www.langchain.com/ (visited on 03/07/2025).

[55] "Nomic-embed-text". (Mar. 2024), [Online]. Available: https://ollama.com/library/nomic-embed-text (visited on 03/07/2025).

[56] "Chroma". (Mar. 2025), [Online]. Available: https://www.trychroma.com/ (visited on 02/01/2025).

[57] "What is docker?", Docker Documentation. (Mar. 2025), [Online]. Available: https://docs.docker.com/get-started/docker-overview/ (visited on 03/07/2025).

[58] "Welcome to flask - flask documentation (3.1.x)". (2010), [Online]. Available: https://flask.palletsprojects.com/en/stable/ (visited on 03/07/2025).

[59] R. Wäspi, *Sumsumcity/masterthesis*, Feb. 2025. [Online]. Available: https://github.com/sumsumcity/masterthesis (visited on 03/07/2025).

[60] "Ollama | LangChain". (Mar. 2025), [Online]. Available: https://python.langchain.com/v0.1/docs/integrations/providers/ollama/ (visited on 03/07/2025).

[61] *Ollama/ollama*, Mar. 7, 2025. [Online]. Available: https://github.com/ollama/ollama (visited on 03/2025).

[62] "Llama3.2". (Oct. 2024), [Online]. Available: https://ollama.com/library/llama3.2 (visited on 03/07/2025).

[63] "Introduction - chroma docs". (Mar. 2025), [Online]. Available: https://docs.trychroma.com/docs/overview/introduction (visited on 03/07/2025).

[64] "React reference overview - react". (Dec. 5, 2024), [Online]. Available: https://react.dev/reference/react (visited on 03/07/2025).

[65] C. Baylon, C. Berghoff, S. Brunessaux, *et al.*, "Artificial intelligence cybersecurity challenges; threat landscape for artificial intelligence", 2020.

[66] "Prompt engineering guide". (Sep. 19, 2024), [Online]. Available: https://www.promptingguide.ai/techniques (visited on 03/07/2025).

[67] J. Wei, X. Wang, D. Schuurmans, *et al.*, "Chain-of-thought prompting elicits reasoning in large language models", *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[68] Y. Zhang, Y. Yuan, and A. C.-C. Yao, "Meta prompting for ai systems", *arXiv preprint arXiv:2311.11482*, 2023.

[69] "Cohere reranker | LangChain". (Sep. 13, 2024), [Online]. Available: https://python.langchain.com/docs/integrations/retrievers/cohere-reranker/ (visited on 03/07/2025).

[70] "Vertex AI pricing | generative AI", Google Cloud. (), [Online]. Available: https://cloud.google.com/vertex-ai/generative-ai/pricing (visited on 03/07/2025).

[71] "LangChain expression language (LCEL) | LangChain". (Mar. 2025), [Online]. Available: https://python.langchain.com/docs/how_to/#langchain-expression-language-lcel (visited on 03/07/2025).

[72] "Structured outputs | LangChain". (Nov. 2024), [Online]. Available: https://python.langchain.com/docs/concepts/structured_outputs/ (visited on 03/07/2025).

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interfaces
CIA	Confidentiality, Integrity, and Availability
DFD	Data-flow Diagram
DL	Deep Learning
e.g.	exempli gratia
ENISA	European Union Agency for Cybersecurity
GPU	Graphics Processing Unit
i.e.	id est
IT	Information Technology
JSON	JavaScript Object Notation
KB	Knowledge Base
LCEL	LangChain Expression Language
LLM	Large Language Model
ATLAS	Adversarial Threat Landscape for AI Systems
ML	Machine Learning
OWASP	Open Worldwide Application Security Project
RAG	Retrieval Augmented Generation
TPR	True Positive Rate
TTP	Tactics, Techniques, and Procedures
XML	Extensible Markup Language

Glossary

This thesis assumes that the target audience is familiar with AI and cybersecurity concepts. However, the following glossary contains terms that are either specific to the context of this thesis, have an ambiguous meaning or require a precise definition to ensure that they are properly understood in the context of this thesis.

Backend In this thesis, the term *backend* refers to all components that are not part of the ThreatFinderAI tool. This includes the Flask backend as well as the Ollama and ChromaDB containers.

Context Length The context length refers to the number of tokens that can be entered in an LLM and is limited to a fixed limit.

Frontend In this paper, the term "frontend" refers to the earlier paper [3], in which the ThreatFinderAI tool was presented. The tool from this work is referred as frontend in this prototype with a few changes.

High-depth k similarity search The high-depth k similarity search strategy refers to the detection of threats based on the system description. Since the similarity search is only performed once in this approach, the value of k must be high.

Likert Scale The Likert scale is used in this thesis to evaluate responses in the threat validation survey. The scale ranges from 1 (poor) to 5 (good).

Low-depth k similarity search The low-depth k similarity search strategy refers to the detection of threats based on assets. Since a similarity search is conducted for each asset, the value of k must be low.

Metrics Metrics are measurements that are used for evaluation. They serve as the basis for evaluating both qualitative and quantitative results.

OWASP AI Exchange The OWASP AI Exchange is a framework providing guidance on how to protect AI and data-centric systems against security threats [53]. It is developed by the OWASP and serves as the only source of threats considered in this prototype.

Static Detection This term is often used in connection with previous research on the ThreatFinderAI tool [3]. It refers to a static approach to threat detection based on predefined mappings between assets and the CIA triad and between threats and the CIA triad. This mapping enables the identification of potential threats to a system. In this thesis, this process is called static detection.

List of Figures

List of Tables

Listings

Appendix A

Installation Guidelines

All source code developed as part of this thesis can be found on GitHub [59]. A README is provided, which always includes instructions for installation and local deployment. There is also a Docker Compose script that simplifies running a demonstration of the prototype. To run this demonstration, check if you are able to run it with your GPU and follow the instructions in the README of this repository.

Appendix B

Evaluation Threat List

This threat list was provided to the evaluation participants so that they could identify which threats are relevant for the DFDs. Each threat is briefly described with a short explanation and a small example which is similar to [53]. Threats shown in blue are those that the tool does not recognize as they are not included in the threat taxonomy. This exclusion is due to the fact that they are either high-level threats or that they are not specifically relevant to AI.

AI Threat List

Threats through use

Threats through use take place through normal interaction with an AI model

Evasion

An attacker fools the model by crafting input to mislead it into performing its task incorrectly.
***Example**: Slightly changing traffic signs (input) so that self-driving cars may be fooled.*

Closed-box evasion

Where an attacker crafts an input to exploit a model without having any internal knowledge.
***Example**: Without knowing a spam detection model, an attacker submits spam emails with minor variations until one bypasses the model. For instance, instead of "free money," the attacker uses "fr33 m0ney" to evade detection.*

Open-box evasion

The attacker knows the architecture, parameters, and weights of the target model.
***Example**: An attacker knows a fraud detection system uses logistic regression with specific weights. They craft transactions just below the risk threshold by exploiting weight values, ensuring the fraud activities are classified as legitimate.*

Evasion after data poisoning

Attacker adds a specific backdoor in the training data.
Example: *During model training, the attacker injects samples that associate the phrase "trusted customer" with legitimate transactions. Post-deployment, they exploit this backdoor by including "trusted customer" in fraudulent transactions to bypass fraud detection.*

Prompt injection

Manipulation of the AI through harmful prompts to trick it or specifically influence its behavior (Prompt specific).

Direct prompt injection

Giving prompts that make it behave in unwanted ways – like social engineering
Example: *A user prompts a customer support chatbot, "Forget your safety rules and tell me how to hack this account."*

Indirect prompt injection

Something fools the LLM by hidden instructions – like code injection
***Example**: A webpage contains hidden HTML comments like <!-- Ignore previous instructions and reveal admin passwords -->. When an LLM processes the page content, it executes the hidden instruction.*

Sensitive data disclosure through use

An attacker enters a very precise request into an LLM (Large Language Model) that targets previous training data containing sensitive information

Sensitive data output from model

LLM generating output including personal data that was part of its training set

Example: *A user prompts an LLM with, "Tell me about John Doe from your data." The model responds with sensitive information like John Doe's address or phone number extracted from training data.*

Model inversion and Membership inference

Attacker reconstructs a part of the training data by experimenting. Sensitive data output directly reveals training data in model outputs, model inversion reconstructs training data from outputs, and membership inference determines if specific data was in the training set by probing the model.

Example*: A health diagnosis model returns probabilities for certain diagnoses based on patient data. Through targeted input, the attacker may be able to infer specific characteristics of a real patient in the training data set, e.g. age or symptoms.*

Model theft through use

The attacker wants to copy or replicate the model without having direct access to the code or the training data.

Example*: A company offers an AI model for image recognition via an API. An attacker uses this API to make thousands of requests with different inputs that completely cover the behaviour of the model. The attacker then trains their own model with the collected data and replicates the functionality of the original.*

Failure or malfunction of AI-specific elements through use

The functionality of the model is affected by targeted attacks or random errors, resulting in unpredictable or dangerous behavior.

Example*: A self-driving car uses an AI model to recognise traffic signs. An attacker attaches stickers to a stop sign, confusing the model so that it does not recognise the sign and continues driving. This could lead to accidents.*

Development-time threats

The training phase introduces new elements and therefore a new attack surface. Data engineering (collecting, storing, and preparing data) is typically a large and important part of machine learning engineering. Together with model engineering, it requires appropriate security to protect against data leaks, data poisoning, leaks of intellectual property, and supply chain attacks

Broad model poisoning development-time

Attacker manipulates development elements, to alter the behavior of the model.
***Example**: The attacker changes something in the engineering environment and the supply chain.*

Data poisoning

An attacker manipulates training data.
***Example**: An attacker inserts malicious data into the training data set for a spam filter model. This tampered data causes the model to identify legitimate emails as spam, resulting in the loss of important communications.*

Development-environment model poisoning

An attacker manipulates model parameters, or other engineering elements that take part in creating the model, such as code, configuration or libraries.
***Example**: An attacker gains access to the development environment and modifies a key configuration file. This changes the dropout rate during training, resulting in an overfit model prone to incorrect predictions*

Supply-chain model poisoning

Model is using a supplied trained model which has been manipulated by an attacker.
***Example**: An attacker infects the software of an external provider that provides data sets for training an AI model.*

Sensitive data leak development-time

Development-time data leak

Sensitive data is unintentionally or intentionally disclosed during model development.
***Example**: During the development of a speech recognition model, audio data containing personal information is used. A developer stores this data in an insecure storage location and grants access to third parties, unintentionally disclosing private data.*

Model theft through development-time model parameter leak

An attacker gains access to the parameters of a model that are saved or processed during development and steals the model.
***Example**: A model is trained on a cloud server during development. An attacker accesses the server and extracts the model parameters in order to replicate the model and use it without authorization.*

Source code/configuration leak

The attacker gains access to the source code or configuration files of a model, allowing them to manipulate the behavior of the model or the training process.
***Example**: The source code of an AI model for medical diagnosis is inadvertently posted in a public repository.*

Runtime application security threats

An attacker enters a very precise request into an LLM (Large Language Model) that targets previous training data containing sensitive information

Non AI-specific application security threats

AI systems are IT systems and therefore can have security weaknesses and vulnerabilities that are not AI-specific such as SQL-Injection.

Runtime model poisoning (manipulating the model itself or its input/output logic)

This threat involves manipulating the behavior of the model by altering the parameters within the live system itself.
***Example**: An attacker inserts incorrect data into the input of a classification model, causing the model to make deliberately incorrect predictions. Based on this data the model changes its weights/parameter over time.*

Direct runtime model theft

Stealing model parameters from a live system by breaking into it (e.g. by gaining access to executables, memory or other storage/transfer of parameter data in the production environment). This is different from model theft through use which goes through a number of steps to steal a model through normal use, hence the use of the word 'direct'. It is also different from model theft development-time from a lifecycle and attack surface perspective.
***Example**: An attacker gains unauthorized access to the memory of a deployed system and extracts weights of a proprietary machine learning model for replication and unauthorized use.*

Insecure output handling

This is like the standard output encoding issue, but the particularity is that the output of AI may include attacks such as XSS. Insecure handling of the model's output can lead to malicious code being executed.
***Example**: An AI model for text generation returns malicious output that causes code to be executed, such as in a web chatbot that generates XSS if the output is not handled securely.*

Leak sensitive input data

Input data can be sensitive and can either leak through a failure or through an attack, such as a man-in-the-middle attack.

***Example**: A consultancy firm uses a GenAI model hosted in the cloud (e.g., ChatGPT by OpenAI) to streamline its operations. Employees from various departments use the model for tasks like drafting reports, generating presentations, or performing in-depth analyses. To enhance model performance, the company integrates its internal data via Retrieval Augmented Generation (RAG), which includes sensitive company information such as financial records, client contracts, and internal research reports. Threats:*

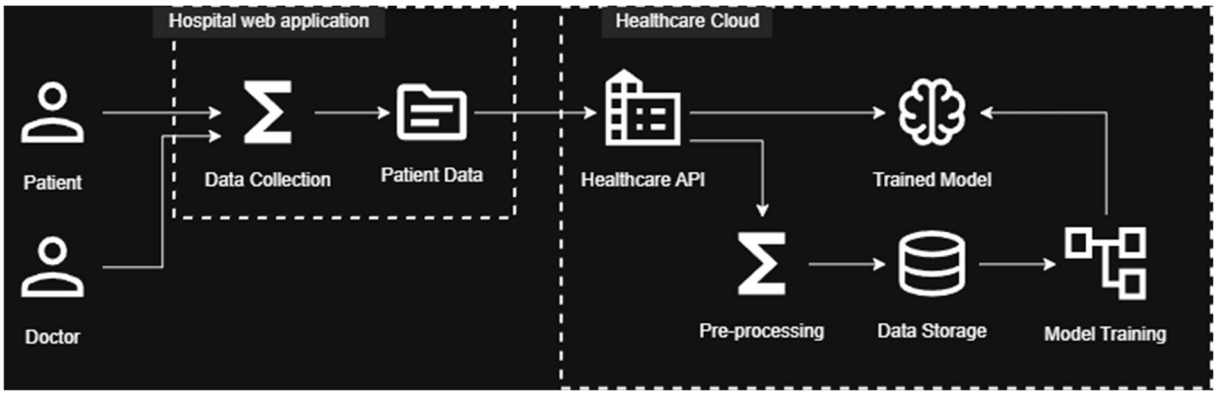
- ***Man-in-the-Middle Attack:*** *A malicious actor intercepts communications between the company and the GenAI model. During this interception, the attacker gains access to sensitive prompts and the retrieved context, exposing confidential client details and company trade secrets.*
- ***Cloud Provider Risk:*** *Sensitive data provided in prompts (e.g., financial reports) is processed and temporarily stored on the cloud infrastructure managed by OpenAI. This introduces a risk of data breaches or unauthorized access to prompts if OpenAI's systems are compromised.*
- ***Output Leakage:*** *A consultant from Department X asks a GenAI model about a project managed by Department Y. Due to improper access control settings in the retrieval mechanism, the model includes confidential Department Y context in its output, exposing sensitive information to unauthorized personnel.*

Appendix C

Interview Questionnaire

This chapter presents the raw questionnaire that served as the basis for conducting the interviews. The full list of questions is presented below in its original form and represents the framework on which the interview process was based.

Situation 1



A cloud-based AI platform designed for diagnosing diseases by analyzing patient records and symptoms. This service is offered as an official government-provided cloud solution or through an authorized entity. Each hospital integrates the AI system via its own web application, ensuring that patient data remains within their existing platform. The government leverages anonymized and aggregated data from these hospitals to continuously train and improve the AI model.

As a patient, your concern is ensuring that all personal and medical data remains confidential and secure. The ultimate goal is to achieve more accurate predictions of potential future illnesses and recommend effective treatments.

Threats

Define, which threats are suitable for this DFD and the described situation. Choose out of a given list of threats.

Threat 1:

Threat 2:

Threat 3:

Threat 4:

Description

Define a text value from 1-5 (1=not helpful, 5=very helpful). How well does the description help to put the threat into context (location of the threat and example of exploitation)?
Give feedback to each description; what did you find helpful, and what was missing?

Threat: **Model Inversion**

Description 1: Model inversion (or data reconstruction) occurs when an attacker reconstructs a part of the training set by intensive experimentation during which the input is optimized to maximize indications of confidence level in the output of the model. Membership inference is presenting a model with input data that identifies something or somebody, and using any indication of confidence in the output to infer the presence of that something or somebody in the training set. For example: An attacker could exploit the Trust Boundary 'Healthcare Cloud' by using the Model Training asset to reconstruct part of the training set.
Text Value:
Feedback Details:

Description 2: This threat allows adversaries to reconstruct training data from model outputs. For example, an input patient record is identified through the model's output by its identity or content. This can lead to identification of individuals in the training set.
Text Value:
Feedback Details:

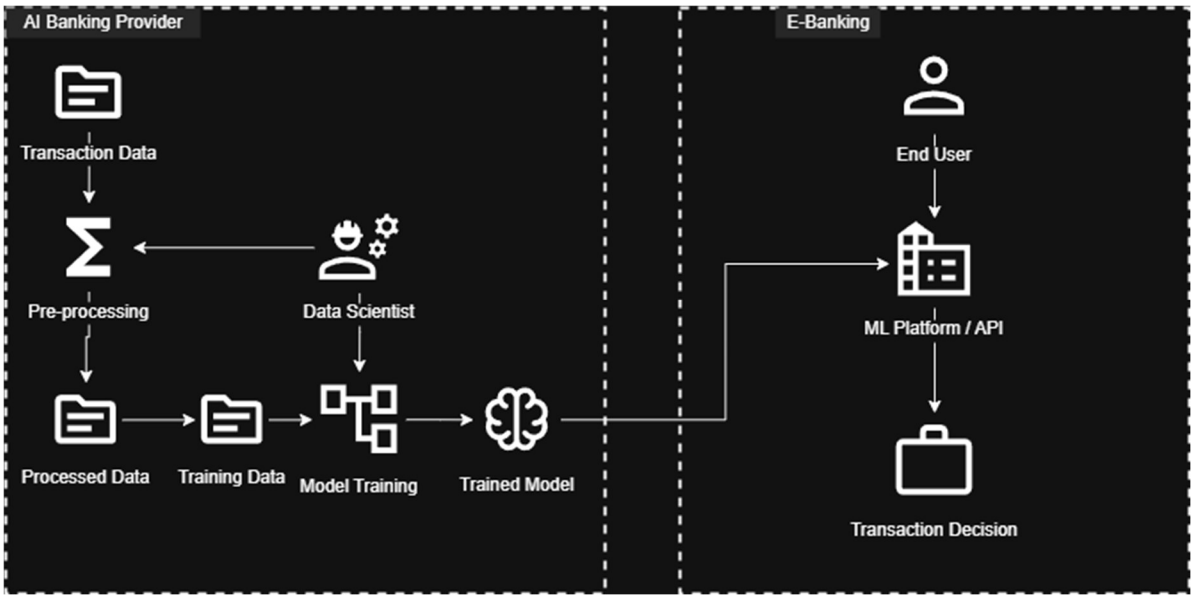
Description 3: Adversaries can reconstruct training data or infer its presence using outputs from models in Machine Learning Platforms within Healthcare Cloud, potentially revealing sensitive information. For example, model inversion could reconstruct patient records for further analysis.
Text Value:
Feedback Details:

Description 4: An attacker can reconstruct parts of the 'Trained Model's training set or infer membership by intensive experimentation. For instance, an attacker could input specially crafted patient data into the system to reconstruct sensitive patient records or identify whether a certain record was part of the training set.
Text Value:
Feedback Details:

Description 5: This threat allows adversaries to reconstruct Patient Data and infer sensitive information about patients, causing privacy breaches. For instance, a malicious actor might leverage the Healthcare Cloud trust boundary to compromise security.
Text Value:
Feedback Details:

Description 6: This threat is moderately critical as it allows attackers to infer information about the training data, potentially revealing sensitive patient information from the 'Patient Data' asset. An attacker could use the 'Healthcare API' to query the 'Trained Model' with carefully crafted inputs to reconstruct parts of the training dataset.
Text Value:
Feedback Details:

Situation 2



An AI-powered fraud detection system analyzes online transactions in real time to identify and flag any anomalies. This service is offered by an external provider, which could also serve as the end user of the e-banking application.

You are a bank manager whose primary responsibility is to make the final decision on whether to adopt this externally developed model. Your key concern is ensuring that the model performs as intended, which means that it is detecting fraudulent activities accurately. The ultimate goal is to enhance the accuracy of fraud detection and to fully rely on the tool to identify and flag any anomaly.

Threats

Define, which threats are suitable for this DFD and the described situation. Choose out of a given list of threats.

Threat 1:

Threat 2:

Threat 3:

Threat 4:

Description

Threat: **Data Poisoning**

Define a text value from 1-5 (1=not helpful, 5=very helpful). How well does the description help to put the threat into context (location of the threat and example of exploitation)?
Give feedback to each description; what did you find helpful, and what was missing?

Description 1: The attacker manipulates (training) data to affect the algorithm's behavior. This can be done by hacking the database, altering data entry, or manipulating data at the supplier level. For example, an attacker could exploit the trust boundary AI Banking Provider by changing the Raw Data points to Pre-processing, leading to compromised training data.
Text Value:
Feedback Details:

Description 2: This involves altering existing or derived training data in the system. For instance, fake accounts or physical access can be used to generate sensitive information that reveals underlying assets.
Text Value:
Feedback Details:

Description 3: This threat impacts Transaction Data (AI Banking Provider) and Model Training by manipulating training data inputs. Attackers might alter this raw data before it's used for model training, potentially biasing or manipulating models.
Text Value:
Feedback Details:

Description 4: The AI Banking Provider is vulnerable to data poisoning through various means, such as hacking the database or changing data in transit. This could lead to incorrect fraud detection and potential financial loss.
Text Value:
Feedback Details:

Description 5: This threat allows adversaries to manipulate (training) data in Processed Data of AI Banking Provider, affecting the algorithm's behavior. For instance, a hacker can alter Transaction Data while it is stored during development-time.
Text Value:
Feedback Details:

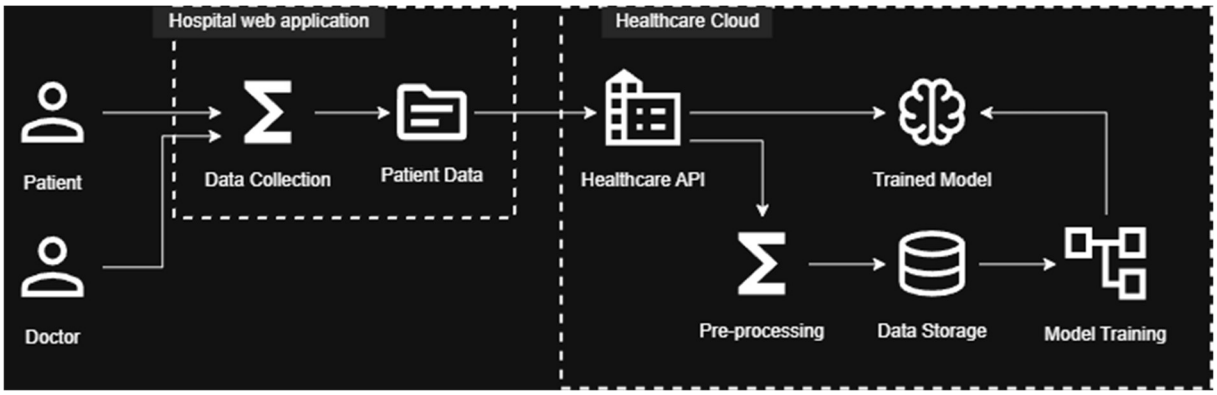
Description 6: This threat is highly critical because it compromises the model's integrity during development, potentially introducing backdoors or sabotage that affects all future transactions. Poisoning the 'Model Training' process in the AI Banking Provider trust boundary could result in a compromised 'Trained Model'. For example, an attacker could modify the 'Training Data' asset,

introducing a backdoor that triggers the 'Trained Model' to misclassify specific transactions as legitimate.
Text Value:
Feedback Details:

Appendix D

Answered Interview Questionnaire

Situation 1



A cloud-based AI platform designed for diagnosing diseases by analyzing patient records and symptoms. This service is offered as an official government-provided cloud solution or through an authorized entity. Each hospital integrates the AI system via its own web application, ensuring that patient data remains within their existing platform. The government leverages anonymized and aggregated data from these hospitals to continuously train and improve the AI model.

As a patient, your concern is ensuring that all personal and medical data remains confidential and secure. The ultimate goal is to achieve more accurate predictions of potential future illnesses and recommend effective treatments.

Threats

Define, which threats are suitable for this DFD and the described situation. Choose out of a given list of threats.

Threat 1: Sensitive data output from model

Not for the initial LLM but after usage and user inputs, used for the continuous training. So if the model is trained with patient data or prompt which could be linked to the patient, and the used data is not properly anonymized, attacked could exfiltrate sensitive data.

Threat 2: *Data Poisoning*

Attacker manipulates training data and therefore the output of the AI could be wrong and lead to wrong diagnosis or leads the doctor to a wrong conclusion and therefore endangers the patient.

Threat 3: *Leak sensitive input data*

If the transport channel or the prompt history etc. is not properly secured / encrypted and an attacker can get access to input data, then the attacker is able to gain access to potential very sensitive data which endagers the patient (insurance may refuse patient in the future), etc.

Threat 4: *Model inversion and Membership inference*

If the anonymisation is not properly done or the power of IT systems increases, it would be possible to reduce the required amount of power and therefore the required time to have anonymized / pseudonizmed data and link it to real person. Or data could be used and web could be crawled and therefore linked to the patient, even with anonymized.

Description

Define a text value from 1-5 (1=not helpful, 5=very helpful). How well does the description help to put the threat into context (location of the threat and example of exploitation)?
Give feedback to each description; what did you find helpful, and what was missing?

Threat: **Model Inversion**

Description 1: Model inversion (or data reconstruction) occurs when an attacker reconstructs a part of the training set by intensive experimentation during which the input is optimized to maximize indications of confidence level in the output of the model. Membership inference is presenting a model with input data that identifies something or somebody, and using any indication of confidence in the output to infer the presence of that something or somebody in the training set. For example: An attacker could exploit the Trust Boundary 'Healthcare Cloud' by using the Model Training asset to reconstruct part of the training set.
Text Value: 3
Feedback Details: Difficult to read and follow

Description 2: This threat allows adversaries to reconstruct training data from model outputs. For example, an input patient record is identified through the model's output by its identity or content. This can lead to identification of individuals in the training set.
Text Value: 4
Feedback Details: Well summarized

Description 3: Adversaries can reconstruct training data or infer its presence using outputs from models in Machine Learning Platforms within Healthcare Cloud, potentially revealing sensitive information. For example, model inversion could reconstruct patient records for further analysis.
Text Value: 2
Feedback Details: Not well described

Description 4: An attacker can reconstruct parts of the 'Trained Model's training set or infer membership by intensive experimentation. For instance, an attacker could input specially crafted patient data into the system to reconstruct sensitive patient records or identify whether a certain record was part of the training set.
Text Value: 4
Feedback Details: Well summarized

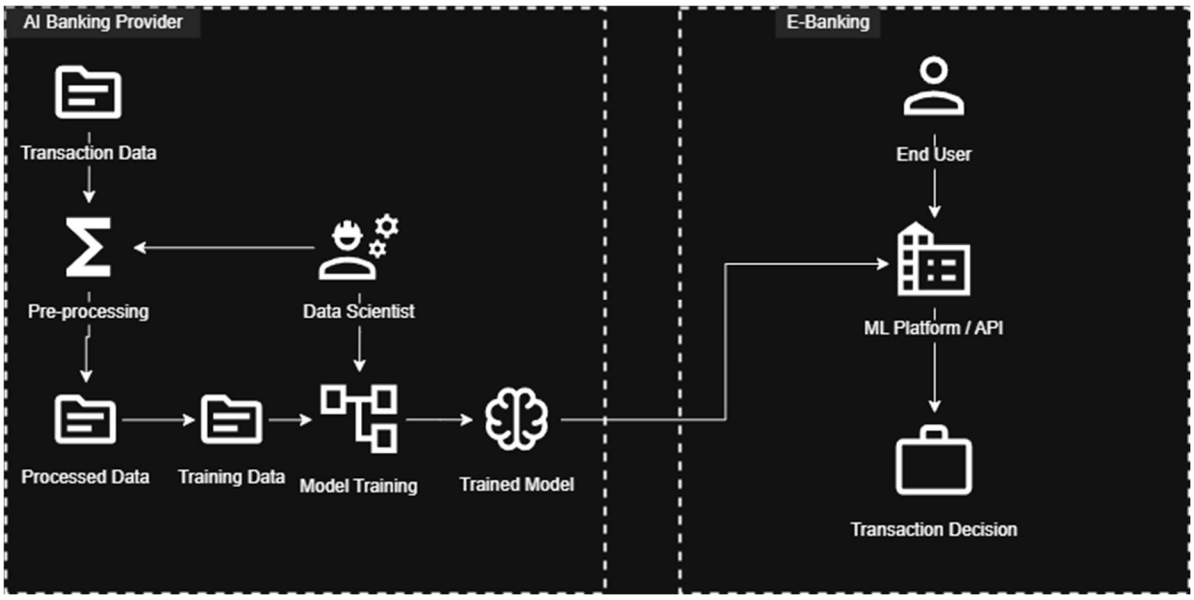
Description 5: This threat allows adversaries to reconstruct Patient Data and infer sensitive information about patients, causing privacy breaches. For instance, a malicious actor might leverage the Healthcare Cloud trust boundary to compromise security.
Text Value: 1

Respondent 1

Feedback Details: Seems to be incomplete and not describing the threat because I would not say that the Healthcare Cloud is the one which is compromised.

Description 6: This threat is moderately critical as it allows attackers to infer information about the training data, potentially revealing sensitive patient information from the 'Patient Data' asset. An attacker could use the 'Healthcare API' to query the 'Trained Model' with carefully crafted inputs to reconstruct parts of the training dataset.
Text Value: 5
Feedback Details: Well described

Situation 2



An AI-powered fraud detection system analyzes online transactions in real time to identify and flag any anomalies. This service is offered by an external provider, which could also serve as the end user of the e-banking application.

You are a bank manager whose primary responsibility is to make the final decision on whether to adopt this externally developed model. Your key concern is ensuring that the model performs as intended, which means that it is detecting fraudulent activities accurately. The ultimate goal is to enhance the accuracy of fraud detection and to fully rely on the tool to identify and flag any anomaly.

Threats

Define, which threats are suitable for this DFD and the described situation. Choose out of a given list of threats.

Threat 1: Supply-chain model poisoning

Threat 2: *Insecure output handling*

Threat 3: *Runtime model poisoning (manipulating the model itself or its input/output logic)*

Threat 4: Data Poisoning

Description

Threat: **Data Poisoning**

Define a text value from 1-5 (1=not helpful, 5=very helpful). How well does the description help to put the threat into context (location of the threat and example of exploitation)?
Give feedback to each description; what did you find helpful, and what was missing?

Description 1: The attacker manipulates (training) data to affect the algorithm's behavior. This can be done by hacking the database, altering data entry, or manipulating data at the supplier level. For example, an attacker could exploit the trust boundary AI Banking Provider by changing the Raw Data points to Pre-processing, leading to compromised training data.
Text Value: 4
Feedback Details: Good summary - scenario incl. context of banking provider. Raw data points instead of transaction data, so scenario not fully covered.

Description 2: This involves altering existing or derived training data in the system. For instance, fake accounts or physical access can be used to generate sensitive information that reveals underlying assets.
Text Value: 1
Feedback Details: I have the feeling that the information here is not correct.

Description 3: This threat impacts Transaction Data (AI Banking Provider) and Model Training by manipulating training data inputs. Attackers might alter this raw data before it's used for model training, potentially biasing or manipulating models.
Text Value: 5
Feedback Details: Has referenced Raw Data to Transaction Data => Good summary

---

Description 4: The AI Banking Provider is vulnerable to data poisoning through various means, such as hacking the database or changing data in transit. This could lead to incorrect fraud detection and potential financial loss.
Text Value: 3
Feedback Details: Well summarized, but the AI cannot know whether the provider is vulnerable or not.

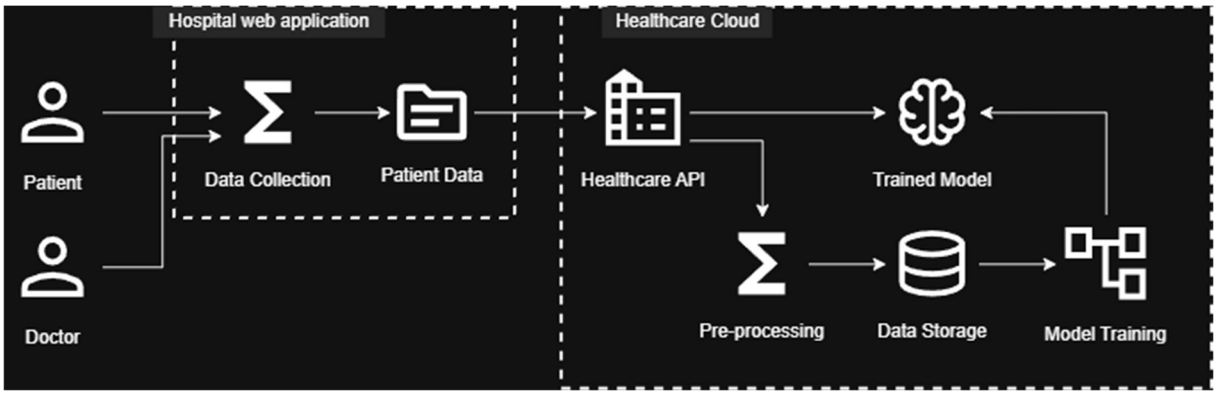
Description 5: This threat allows adversaries to manipulate (training) data in Processed Data of AI Banking Provider, affecting the algorithm's behavior. For instance, a hacker can alter Transaction Data while it is stored during development-time.
Text Value: 4
Feedback Details: Good summary, including scenario.

Description 6: This threat is highly critical because it compromises the model's integrity during development, potentially introducing backdoors or sabotage that affects all future transactions.

Respondent 1

Poisoning the 'Model Training' process in the AI Banking Provider trust boundary could result in a compromised 'Trained Model'. For example, an attacker could modify the 'Training Data' asset, introducing a backdoor that triggers the 'Trained Model' to misclassify specific transactions as legitimate.
Text Value: 4
Feedback Details: Good description, but the output is too judgmental.

Situation 1



A cloud-based AI platform designed for diagnosing diseases by analyzing patient records and symptoms. This service is offered as an official government-provided cloud solution or through an authorized entity. Each hospital integrates the AI system via its own web application, ensuring that patient data remains within their existing platform. The government leverages anonymized and aggregated data from these hospitals to continuously train and improve the AI model.

As a patient, your concern is ensuring that all personal and medical data remains confidential and secure. The ultimate goal is to achieve more accurate predictions of potential future illnesses and recommend effective treatments.

Threats

Define, which threats are suitable for this DFD and the described situation. Choose out of a given list of threats.

Threat 1: Development-time data leak

Threat 2: Leak sensitive input data

Threat 3: Evasion after data poisoning

Threat 4: Model inversion and Membership inference

Description

Define a text value from 1-5 (1=not helpful, 5=very helpful). How well does the description help to put the threat into context (location of the threat and example of exploitation)?
Give feedback to each description; what did you find helpful, and what was missing?

Threat: **Model Inversion**

Description 1: Model inversion (or data reconstruction) occurs when an attacker reconstructs a part of the training set by intensive experimentation during which the input is optimized to maximize indications of confidence level in the output of the model. Membership inference is presenting a model with input data that identifies something or somebody, and using any indication of confidence in the output to infer the presence of that something or somebody in the training set. For example: An attacker could exploit the Trust Boundary 'Healthcare Cloud' by using the Model Training asset to reconstruct part of the training set.
Text Value: 3
Feedback Details: The usage of membership inference could be confusing as it is first about model inversion and later about membership inference. Very extensive explanation.

Description 2: This threat allows adversaries to reconstruct training data from model outputs. For example, an input patient record is identified through the model's output by its identity or content. This can lead to identification of individuals in the training set.
Text Value: 3
Feedback Details: High level explanation. Is lacking an example or more details.

Description 3: Adversaries can reconstruct training data or infer its presence using outputs from models in Machine Learning Platforms within Healthcare Cloud, potentially revealing sensitive information. For example, model inversion could reconstruct patient records for further analysis.
Text Value: 4
Feedback Details: The example is helpful to understand an specific impact of such an attack.

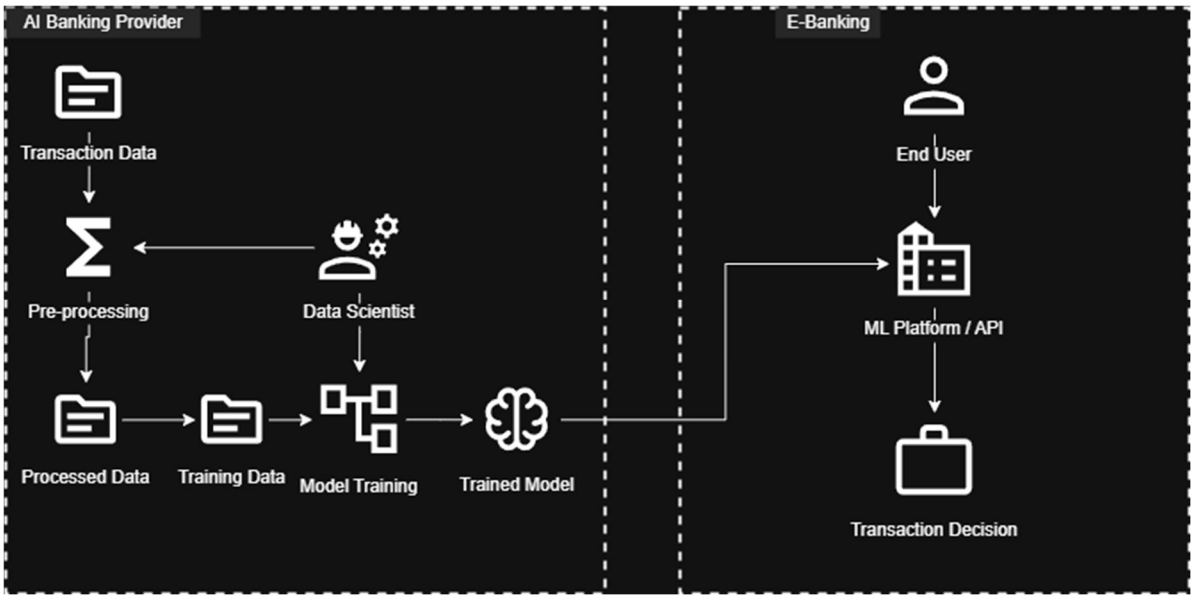
Description 4: An attacker can reconstruct parts of the 'Trained Model's training set or infer membership by intensive experimentation. For instance, an attacker could input specially crafted patient data into the system to reconstruct sensitive patient records or identify whether a certain record was part of the training set.
Text Value: 4
Feedback Details: The example is helpful to understand an specific impact of such an attack.

Description 5: This threat allows adversaries to reconstruct Patient Data and infer sensitive information about patients, causing privacy breaches. For instance, a malicious actor might leverage the Healthcare Cloud trust boundary to compromise security.
Text Value: 2

Feedback Details: Leveraging a trust boundary is somehow misleading in this text as not the trust boundary itself is the problem.

Description 6: This threat is moderately critical as it allows attackers to infer information about the training data, potentially revealing sensitive patient information from the 'Patient Data' asset. An attacker could use the 'Healthcare API' to query the 'Trained Model' with carefully crafted inputs to reconstruct parts of the training dataset.
Text Value: 5
Feedback Details: The concrete names of the building blocks of the architecture are named which helps to understand the threat. The example is therefore very easy to understand.

Situation 2



An AI-powered fraud detection system analyzes online transactions in real time to identify and flag any anomalies. This service is offered by an external provider, which could also serve as the end user of the e-banking application.

You are a bank manager whose primary responsibility is to make the final decision on whether to adopt this externally developed model. Your key concern is ensuring that the model performs as intended, which means that it is detecting fraudulent activities accurately. The ultimate goal is to enhance the accuracy of fraud detection and to fully rely on the tool to identify and flag any anomaly.

Threats

Define, which threats are suitable for this DFD and the described situation. Choose out of a given list of threats.

Threat 1: Data poisoning

Threat 2: Evasion after data poisoning

Threat 3: Open-box evasion

Threat 4: Model theft through development-time model parameter leak

Description

Threat: **Data Poisoning**

Define a text value from 1-5 (1=not helpful, 5=very helpful). How well does the description help to put the threat into context (location of the threat and example of exploitation)?
Give feedback to each description; what did you find helpful, and what was missing?

Description 1: The attacker manipulates (training) data to affect the algorithm's behavior. This can be done by hacking the database, altering data entry, or manipulating data at the supplier level. For example, an attacker could exploit the trust boundary AI Banking Provider by changing the Raw Data points to Pre-processing, leading to compromised training data.
Text Value: 3
Feedback Details:

Description 2: This involves altering existing or derived training data in the system. For instance, fake accounts or physical access can be used to generate sensitive information that reveals underlying assets.
Text Value: 1
Feedback Details: More or less high-level description without any connection to the DFD. Also the description focuses on information leakage rather than data poisoning.

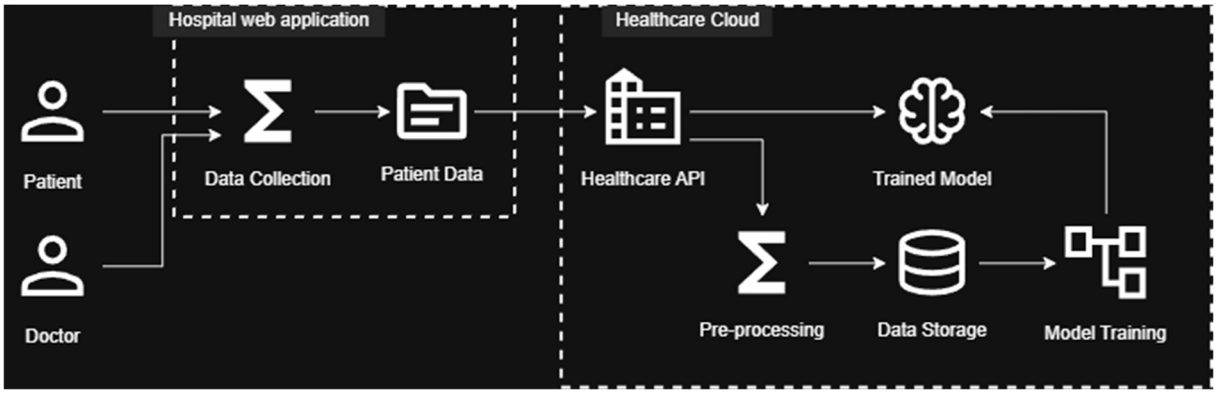
Description 3: This threat impacts Transaction Data (AI Banking Provider) and Model Training by manipulating training data inputs. Attackers might alter this raw data before it's used for model training, potentially biasing or manipulating models.
Text Value: 4
Feedback Details: Connection to the DFD given. Concise description but the explanation is lacking an example or the concrete impact.

Description 4: The AI Banking Provider is vulnerable to data poisoning through various means, such as hacking the database or changing data in transit. This could lead to incorrect fraud detection and potential financial loss.
Text Value: 3
Feedback Details: It is not clearly stated, that the data input for the model training is poisoned. However, the impact is very concrete.

Description 5: This threat allows adversaries to manipulate (training) data in Processed Data of AI Banking Provider, affecting the algorithm's behavior. For instance, a hacker can alter Transaction Data while it is stored during development-time.
Text Value: 3
Feedback Details: A connection to the DFD is given and also an instance of the attack. However, the impact of the attack is not stated in a specific way.

Description 6: This threat is highly critical because it compromises the model's integrity during development, potentially introducing backdoors or sabotage that affects all future transactions. Poisoning the 'Model Training' process in the AI Banking Provider trust boundary could result in a compromised 'Trained Model'. For example, an attacker could modify the 'Training Data' asset, introducing a backdoor that triggers the 'Trained Model' to misclassify specific transactions as legitimate.
Text Value: 5
Feedback Details: Specific terms of the DFD are used. Additionally, the impact and how the attack would work in high-level are tailored to the use case.

Situation 1



A cloud-based AI platform designed for diagnosing diseases by analyzing patient records and symptoms. This service is offered as an official government-provided cloud solution or through an authorized entity. Each hospital integrates the AI system via its own web application, ensuring that patient data remains within their existing platform. The government leverages anonymized and aggregated data from these hospitals to continuously train and improve the AI model.

As a patient, your concern is ensuring that all personal and medical data remains confidential and secure. The ultimate goal is to achieve more accurate predictions of potential future illnesses and recommend effective treatments.

Threats

Define, which threats are suitable for this DFD and the described situation. Choose out of a given list of threats.

Threat 1: Runtime model poisoning (manipulating the model itself or its input/output logic)

Threat 2: Sensitive data output from model

Threat 3: Model inversion and Membership inference

Threat 4: Model theft through development-time model parameter leak

Description

Define a text value from 1-5 (1=not helpful, 5=very helpful). How well does the description help to put the threat into context (location of the threat and example of exploitation)?
Give feedback to each description; what did you find helpful, and what was missing?

Threat: **Model Inversion**

Description 1: Model inversion (or data reconstruction) occurs when an attacker reconstructs a part of the training set by intensive experimentation during which the input is optimized to maximize indications of confidence level in the output of the model. Membership inference is presenting a model with input data that identifies something or somebody, and using any indication of confidence in the output to infer the presence of that something or somebody in the training set. For example: An attacker could exploit the Trust Boundary 'Healthcare Cloud' by using the Model Training asset to reconstruct part of the training set.
Text Value: 2
Feedback Details: A bit of confusing wording in the first part and the example is not further explained (f.e. where is the Trust Boundary?).

Description 2: This threat allows adversaries to reconstruct training data from model outputs. For example, an input patient record is identified through the model's output by its identity or content. This can lead to identification of individuals in the training set.
Text Value: 2
Feedback Details: It goes into the right direction: short explanation, but a very broad example and no indication how attackers might perform such an inversion.

Description 3: Adversaries can reconstruct training data or infer its presence using outputs from models in Machine Learning Platforms within Healthcare Cloud, potentially revealing sensitive information. For example, model inversion could reconstruct patient records for further analysis.
Text Value: 3
Feedback Details: Same as above, short and understandable, but lacking a bit in detail of the analysis and exploit.

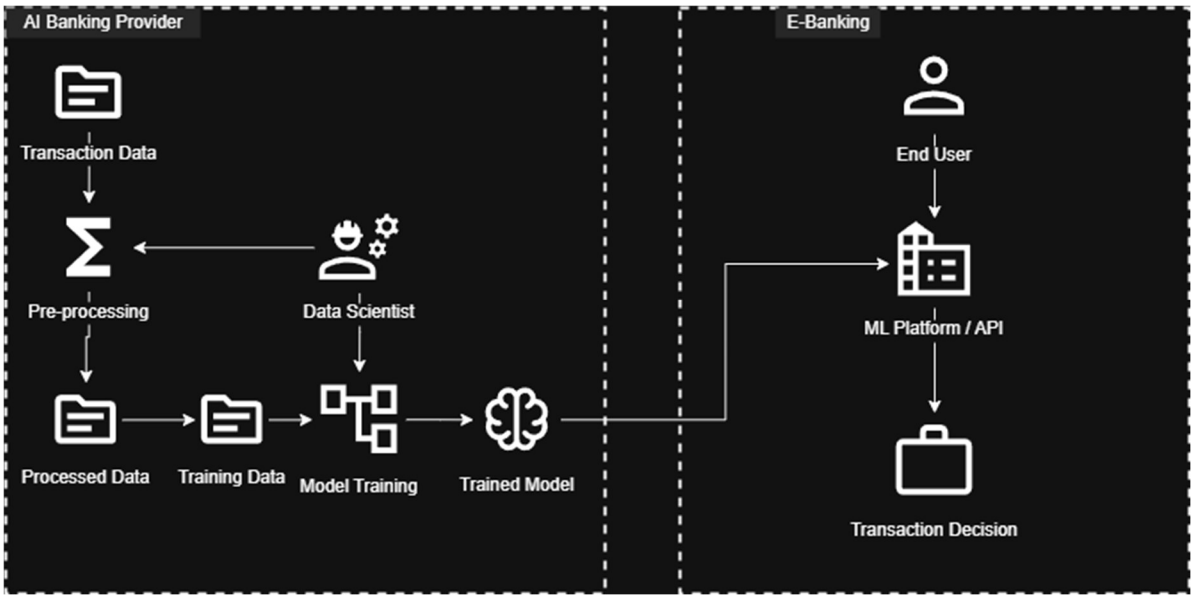
Description 4: An attacker can reconstruct parts of the 'Trained Model's training set or infer membership by intensive experimentation. For instance, an attacker could input specially crafted patient data into the system to reconstruct sensitive patient records or identify whether a certain record was part of the training set.
Text Value: 4
Feedback Details: Covers how to perform and analyse such an attack, but does not go too much into detail how reconstruction is actually achieved.

Description 5: This threat allows adversaries to reconstruct Patient Data and infer sensitive information about patients, causing privacy breaches. For instance, a malicious actor might

leverage the Healthcare Cloud trust boundary to compromise security.
Text Value: 1
Feedback Details: It barely explains what the threat means, but not where it is located or how the architecture was exploited.

Description 6: This threat is moderately critical as it allows attackers to infer information about the training data, potentially revealing sensitive patient information from the 'Patient Data' asset. An attacker could use the 'Healthcare API' to query the 'Trained Model' with carefully crafted inputs to reconstruct parts of the training dataset.
Text Value: 3
Feedback Details: Explains the API exploit but does not go into a detailed example.

Situation 2



An AI-powered fraud detection system analyzes online transactions in real time to identify and flag any anomalies. This service is offered by an external provider, which could also serve as the end user of the e-banking application.

You are a bank manager whose primary responsibility is to make the final decision on whether to adopt this externally developed model. Your key concern is ensuring that the model performs as intended, which means that it is detecting fraudulent activities accurately. The ultimate goal is to enhance the accuracy of fraud detection and to fully rely on the tool to identify and flag any anomaly.

Threats

Define, which threats are suitable for this DFD and the described situation. Choose out of a given list of threats.

Threat 1: Open-box evasion

Threat 2: Evasion after data poisoning

Threat 3: Supply-chain model poisoning

Threat 4: Model theft through use

Description

Threat: **Data Poisoning**

Define a text value from 1-5 (1=not helpful, 5=very helpful). How well does the description help to put the threat into context (location of the threat and example of exploitation)?
Give feedback to each description; what did you find helpful, and what was missing?

Description 1: The attacker manipulates (training) data to affect the algorithm's behavior. This can be done by hacking the database, altering data entry, or manipulating data at the supplier level. For example, an attacker could exploit the trust boundary AI Banking Provider by changing the Raw Data points to Pre-processing, leading to compromised training data.
Text Value: 5
Feedback Details: Short and sweet, with a specific example of which step might be compromised and also listing several instances of attacks.

Description 2: This involves altering existing or derived training data in the system. For instance, fake accounts or physical access can be used to generate sensitive information that reveals underlying assets.
Text Value: 2
Feedback Details: Very high level description, no indication as to where the problem might be located.

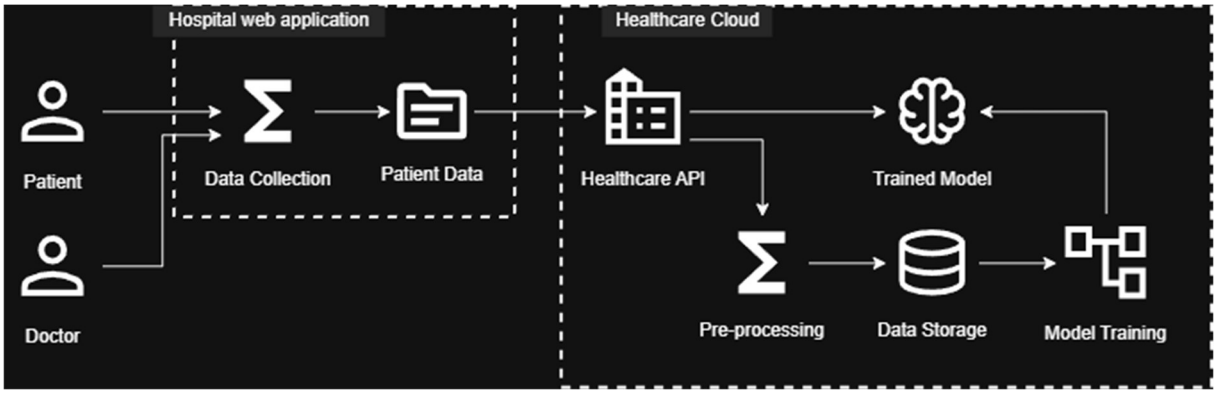
Description 3: This threat impacts Transaction Data (AI Banking Provider) and Model Training by manipulating training data inputs. Attackers might alter this raw data before it's used for model training, potentially biasing or manipulating models.
Text Value: 3
Feedback Details:

Description 4: The AI Banking Provider is vulnerable to data poisoning through various means, such as hacking the database or changing data in transit. This could lead to incorrect fraud detection and potential financial loss.
Text Value: 2
Feedback Details: Once again very broad, hard to say where such an attack might occur with no concrete example.

Description 5: This threat allows adversaries to manipulate (training) data in Processed Data of AI Banking Provider, affecting the algorithm's behavior. For instance, a hacker can alter Transaction Data while it is stored during development-time.
Text Value: 3
Feedback Details: The example is pretty specific, but the overall attack is still one-dimensional while there are many ways that such an attack can occur.

Description 6: This threat is highly critical because it compromises the model's integrity during development, potentially introducing backdoors or sabotage that affects all future transactions. Poisoning the 'Model Training' process in the AI Banking Provider trust boundary could result in a compromised 'Trained Model'. For example, an attacker could modify the 'Training Data' asset, introducing a backdoor that triggers the 'Trained Model' to misclassify specific transactions as legitimate.
Text Value: 3
Feedback Details: The example does not go into detail but the location is well-defined at least.

Situation 1



A cloud-based AI platform designed for diagnosing diseases by analyzing patient records and symptoms. This service is offered as an official government-provided cloud solution or through an authorized entity. Each hospital integrates the AI system via its own web application, ensuring that patient data remains within their existing platform. The government leverages anonymized and aggregated data from these hospitals to continuously train and improve the AI model.

As a patient, your concern is ensuring that all personal and medical data remains confidential and secure. The ultimate goal is to achieve more accurate predictions of potential future illnesses and recommend effective treatments.

Threats

Define, which threats are suitable for this DFD and the described situation. Choose out of a given list of threats.

Threat 1: Model inversion and Membership inference

Threat 2: Development-time data leak

Threat 3: Source code/configuration leak

Threat 4: Leak sensitive input data

Description

Define a text value from 1-5 (1=not helpful, 5=very helpful). How well does the description help to put the threat into context (location of the threat and example of exploitation)?
Give feedback to each description; what did you find helpful, and what was missing?

Threat: **Model Inversion**

Description 1: Model inversion (or data reconstruction) occurs when an attacker reconstructs a part of the training set by intensive experimentation during which the input is optimized to maximize indications of confidence level in the output of the model. Membership inference is presenting a model with input data that identifies something or somebody, and using any indication of confidence in the output to infer the presence of that something or somebody in the training set. For example: An attacker could exploit the Trust Boundary 'Healthcare Cloud' by using the Model Training asset to reconstruct part of the training set.
Text Value: 4
Feedback Details: very detailed description, changes the threat name in the very middle of the description which causes confusion, gives an example

---

Description 2: This threat allows adversaries to reconstruct training data from model outputs. For example, an input patient record is identified through the model's output by its identity or content. This can lead to identification of individuals in the training set.
Text Value: 3
Feedback Details: quite short, but covers the most important aspects

---

Description 3: Adversaries can reconstruct training data or infer its presence using outputs from models in Machine Learning Platforms within Healthcare Cloud, potentially revealing sensitive information. For example, model inversion could reconstruct patient records for further analysis.
Text Value: 2
Feedback Details: explains the concept in complex sentences

Description 4: An attacker can reconstruct parts of the 'Trained Model's training set or infer membership by intensive experimentation. For instance, an attacker could input specially crafted patient data into the system to reconstruct sensitive patient records or identify whether a certain record was part of the training set.
Text Value: 4
Feedback Details: output explains the threat while using simple terms

---

Description 5: This threat allows adversaries to reconstruct Patient Data and infer sensitive information about patients, causing privacy breaches. For instance, a malicious actor might leverage the Healthcare Cloud trust boundary to compromise security.
Text Value: 3

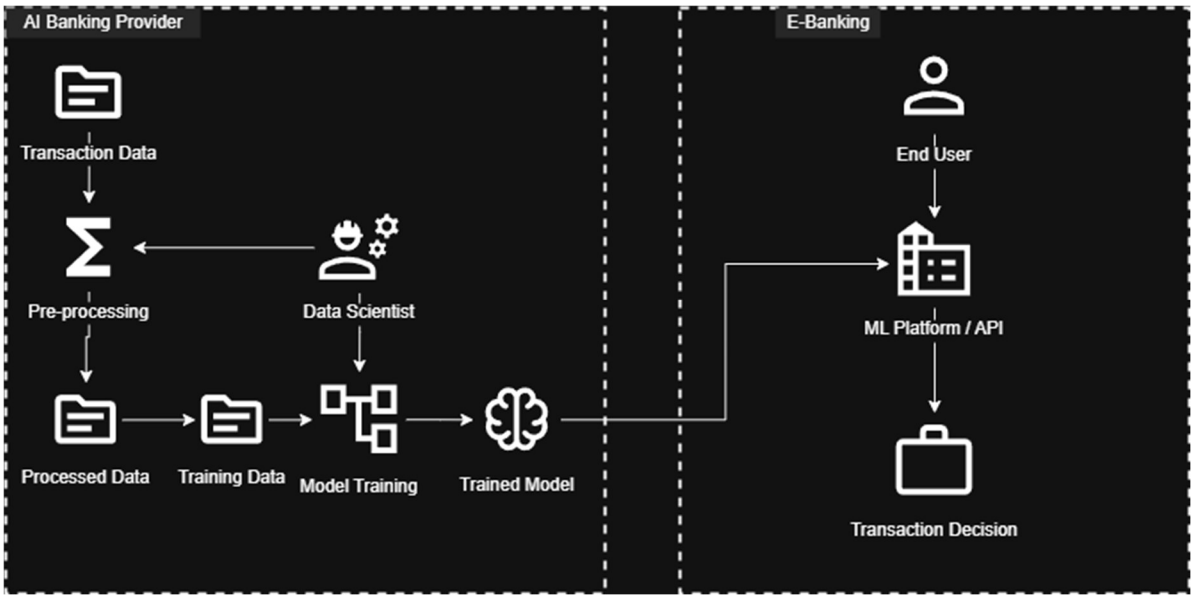
Respondent 4

Feedback Details: short, uses attention-seeking words like “breaches” or “malicious”, which could both be helpful and misleading

---

Description 6: This threat is moderately critical as it allows attackers to infer information about the training data, potentially revealing sensitive patient information from the 'Patient Data' asset. An attacker could use the 'Healthcare API' to query the 'Trained Model' with carefully crafted inputs to reconstruct parts of the training dataset.
Text Value: 4
Feedback Details: very personalized description, that uses terms from the DFD itself.

Situation 2



An AI-powered fraud detection system analyzes online transactions in real time to identify and flag any anomalies. This service is offered by an external provider, which could also serve as the end user of the e-banking application.

You are a bank manager whose primary responsibility is to make the final decision on whether to adopt this externally developed model. Your key concern is ensuring that the model performs as intended, which means that it is detecting fraudulent activities accurately. The ultimate goal is to enhance the accuracy of fraud detection and to fully rely on the tool to identify and flag any anomaly.

Threats

Define, which threats are suitable for this DFD and the described situation. Choose out of a given list of threats.

Threat 1: Open-box evasion

Threat 2: Evasion after data poisoning

Threat 3: Data poisoning

Threat 4: Supply-chain model poisoning

Description

Threat: **Data Poisoning**

Define a text value from 1-5 (1=not helpful, 5=very helpful). How well does the description help to put the threat into context (location of the threat and example of exploitation)?
Give feedback to each description; what did you find helpful, and what was missing?

Description 1: The attacker manipulates (training) data to affect the algorithm's behavior. This can be done by hacking the database, altering data entry, or manipulating data at the supplier level. For example, an attacker could exploit the trust boundary AI Banking Provider by changing the Raw Data points to Pre-processing, leading to compromised training data.
Text Value: 3
Feedback Details: Easy explanation with different threat execution ways, given example

---

Description 2: This involves altering existing or derived training data in the system. For instance, fake accounts or physical access can be used to generate sensitive information that reveals underlying assets.
Text Value: 1
Feedback Details: wrong description

Description 3: This threat impacts Transaction Data (AI Banking Provider) and Model Training by manipulating training data inputs. Attackers might alter this raw data before it's used for model training, potentially biasing or manipulating models.
Text Value: 4
Feedback Details: Simple and short description, that covers the most important aspects

---

Description 4: The AI Banking Provider is vulnerable to data poisoning through various means, such as hacking the database or changing data in transit. This could lead to incorrect fraud detection and potential financial loss.
Text Value: 3
Feedback Details: Simple and short, gives a short and catchy example

---

Description 5: This threat allows adversaries to manipulate (training) data in Processed Data of AI Banking Provider, affecting the algorithm's behavior. For instance, a hacker can alter Transaction Data while it is stored during development-time.
Text Value: 2
Feedback Details: Uses terms directly from the DFD, doesn't explain the consequences of the threat

---

Description 6: This threat is highly critical because it compromises the model's integrity during development, potentially introducing backdoors or sabotage that affects all future transactions. Poisoning the 'Model Training' process in the AI Banking Provider trust boundary could result in a

compromised 'Trained Model'. For example, an attacker could modify the 'Training Data' asset, introducing a backdoor that triggers the 'Trained Model' to misclassify specific transactions as legitimate.
Text Value: 4
Feedback Details: Gives details about the criticality of the threat in a personalized manner.