Offensive Security Tooling for the Automotive Domain

Bachelor Thesis

from the Course of Studies IT-Sicherheit

at the Cooperative State University Baden-Württemberg Ravensburg
Friedrichshafen

by

Dennis Gellert

22.09.2025

Student ID, Course	6018386, TIS22
Company	Continental Automotive Technologies GmbH, 60488 Frankfurt am Main
Supervisor in the Company	Grümer, Patrick, M.Sc
Reviewer	Herr Maier

Confidentiality Statement

Die vorliegende Bachelor Thesis mit dem Titel

Offensive Security Tooling for the Automotive Domain

enthält unternehmensinterne bzw. vertrauliche Informationen der Continental Automotive Technologies GmbH, ist deshalb mit einem Sperrvermerk versehen und wird ausschließlich zu Prüfungszwecken am Studiengang IT-Sicherheit der Dualen Hochschule Baden-Württemberg Ravensburg Friedrichshafen vorgelegt.

Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anders lautende Genehmigung der Ausbildungsstätte (Continental Automotive Technologies GmbH) vorliegt.

Frankfurt am Main, 22.09.2025

Dennis Gellert

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich meine Bachelor Thesis mit dem Thema:

Offensive Security Tooling for the Automotive Domain

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass alle eingereichten Fassungen übereinstimmen.

Frankfurt am Main, 22.09.2025

Dennis Gellert

Abstract

Abstract

Contents

Acronyms

CES	Continental Engineering Services
CAN	Controller Area Network
ECU	Electronic Control Unit
LIN	Local Interconnect Network
MOST	Media Oriented Systems Transport
OEM	Original Equipment Manufacturer
UDS	Unified Diagnostic Services
ABS	Anti Braking System
CRC	Cyclic Redundancy Check
DoS	Denial of Service
IEEE	Institute of Electrical and Electronics Engineers
OBD	On-Board Diagnostics
SAE	Society of Automotive Engineers
Wi-Fi	Wireless Fidelity
LAN	Local Area Network
WEP	Wired Equivalent Privacy
WPA	Wi-Fi Protected Access
ID	Identifier
MAC	Media Access Control
GPU	Graphics Processing Unit

CPU	Central Processing Unit
GUI	Graphical User Interface
TARA	Threat Analysis and Risk Assessments
RAG	Retrieval-Augmented Generation
CVE	Common Vulnerabilities and Exposures
GmbH	Gesellschaft mit beschränkter Haftung
AG	Aktien Gesellschaft
AN	Architecture and Networking
AM	Autonomous Mobility
SAM	Safety and Motion
SCT	Software and Central Technologies
UX	User Experience
CAPEC	Common Attack Pattern Enumeration and Classification
LLM	Large Language Model
STAF	Security Test Automation Framework
ANOVA	analysis of variance
MoRSE	Mixture of RAGs Security Expert
AI	Artificial Intelligence
V2X	Vehicle-to-everything
USB	Universal Serial Bus
SUMS	Software Update Management System
CSMS	Cybersecurity Management System
FAISS	Facebook AI Similarity Search

List of Figures

List of Tables

Listings

1 Introduction

1.1 Motivation

Modern vehicles are no longer isolated systems. They function as connected platforms that continuously interact with cloud services, external interfaces, and other vehicles [1]. With the increasing digitalization of vehicle functions and the growing complexity of automotive electronics, including the presence of up to 150 connected Electronic Control Units (ECUs), the attack surface has expanded significantly [2]. As a result, cybersecurity has become a critical aspect of vehicle development. These developments introduce new risks and make structured, methodical approaches to security analysis essential throughout the development lifecycle.

A key method for identifying potential threats and evaluating their impact is the Threat Analysis and Risk Assessments (TARA), as defined in ISO/SAE 21434 [3]. One central tool in this process is the attack tree: a hierarchical model that breaks down a threat scenario into realistic attacker goals and concrete attack steps. While attack trees are powerful, their manual creation is time-consuming, expertise-driven, and prone to bias or inconsistency.

Recent developments in artificial intelligence, particularly in language models, have made it possible to automate parts of complex analytical tasks. When combined with retrieval methods such as semantic search, these systems can access and apply relevant background knowledge to support decision-making [4]. In the context of cybersecurity, this enables the analysis of regulatory documents, threat intelligence, and existing risk models at a scale that would be difficult to achieve manually. As a result, AI-based approaches offer a promising path toward more efficient and consistent security assessments.

This project builds on that potential. It explores how large language models can be used to generate realistic and valid attack trees that follow strict rules and reflect actual threat behavior. The aim is to develop a retrieval-augmented generation pipeline that supports the TARA process through structured output, consistent formatting, and optional similarity-based guidance from existing models.

1.2 Problem Statement

As automotive systems become increasingly connected and software-driven, the task of identifying and evaluating cybersecurity risks grows in both complexity and importance. Regulatory standards such as ISO/SAE 21434 [3] require a structured TARA process, where one key activity is the construction of attack trees that describe how a threat scenario could realistically occur.

Traditionally, these attack trees are created manually by security analysts. This approach has several well-known limitations:

- **High manual effort** – Analysts must read and interpret complex threat descriptions, identify attacker goals, and model attack paths by hand. This process is labor intensive and does not scale well for large systems.
- **Inconsistent output** – The structure and quality of attack trees often vary between analysts due to differing experience levels and subjective judgment. This affects comparability and traceability.
- **Underuse of historical knowledge** – Existing attack trees, security catalogs, and threat databases are rarely reused systematically, even though they contain valuable context for recurring threat patterns [**THREATGET2023**, 5].

These challenges highlight the need for an approach that supports a more consistent, efficient, and scalable way of constructing attack trees as part of the TARA process.

1.3 Research Objectives and Scope

This thesis explores whether large language models, when combined with retrieval-based techniques, can support the automated generation of structured attack trees as part of the TARA process in the automotive domain. The primary objective is to determine whether such a system can reduce manual modeling effort and improve consistency for cybersecurity analysts.

The study investigates to what extent AI-generated attack trees are reliable, technically meaningful, and structurally valid. A central focus is placed on whether the generated outputs conform to a structured format with hierarchical logic and rating propagation. The Evaluation of Retrieval-Augmented Generation (RAG) focuses on whether relevant prior knowledge such as Common Vulnerabilities and Exposures (CVE), threat catalogs, and existing attack trees is meaningfully reflected in the generated outputs.

Although integration into Ansys Medini Analyze was considered, the tool currently does not support importing or exporting attack trees directly. For this reason, integration is not part of the project scope. However, the structure of the generated trees is designed to reflect the way attack trees are typically modeled, allowing for potential future compatibility. The focus remains on generation and structural consistency rather than toolchain embedding. Medini Analyze was chosen as a reference because it is the tool used within Continental and widely adapted across the automotive industry for model-based safety and security analysis.

To guide this research, the following questions are considered:

- Can large language models, supported by retrieval techniques, be used to generate structured attack trees that follow predefined modeling rules?
- How consistent and realistic are the generated attack trees in terms of structure, node types, rating values, and logical propagation?
- Does retrieved context such as threat catalogs, CVEs, or existing attack trees influence the structure and content of the generated output?

1.4 Project Environment

1.4.1 Continental

Continental is a globally operating company in the field of automotive technology and innovation. Founded in 1871 as a stock corporation in Hannover, the company initially specialized in trading caoutchouc and gutta-percha. By 1892, it became the first German company to manufacture pneumatic bicycle tires and expanded to automotive tires in 1898. The introduction of treaded tires followed in 1904 [6].

The acquisition of Alfred Teves GmbH in 1998 strengthened Continental's capabilities in brake systems and laid the foundation for what would later become Continental Automotive Technologies GmbH [6]. In 2022, the company was restructured under the name Continental AG.

Continental is structured into four main divisions: Automotive, Tires, ContiTech, and Contract Manufacturing, which are further divided into 16 business areas as of 2024. The Automotive division focuses on technologies related to safety, braking, motion control, and automated driving. It consists of five business units: Architecture and Networking (AN), Autonomous Mobility (AM), Safety and Motion (SAM), Software and Central Technologies (SCT), and User Experience (UX).

As a Tier-One supplier, Continental provides a broad product portfolio, including brake systems, radars, sensors, airbag control units, and display devices, serving both the automotive market and adjacent sectors.

1.4.2 Continental Engineering Services

Continental Engineering Services (CES), a business unit within the SCT segment, was established in 2006 to offer specialized engineering services. While its primary focus is on automotive systems, CES also supports industries such as aerospace, rail, marine, and agricultural technology [7].

With over 2,200 employees in 24 locations worldwide, CES combines flexibility with access to Continental's broader technical infrastructure. Its scope includes both adaptation of existing technologies and the development of customer-specific innovations for small series and industrial applications.

Within CES, the department "Systems Engineering – Product Cybersecurity and Privacy" plays a central role in the secure development of vehicle components. It is one of four global competence teams under the Systems Engineering division, working alongside groups for system architecture, homologation, and product safety.

The cybersecurity team is responsible for ensuring that developed products meet internal and external security requirements, including support for risk analysis, testing, and inspection. As part of this, the department explores the potential for automation in areas such as penetration testing and security validation to improve efficiency and reduce manual effort in early development phases.

1.5 Structure of the Thesis

The thesis is divided into eight chapters that document the development, implementation, and evaluation of a system for automating the generation of attack trees using retrieval-augmented generation (RAG) techniques. Chapter 1 introduces the motivation, outlines the problem statement, and defines the research objectives that guide this work. Chapter 2 provides a literature review of existing research on TARA methodologies, attack tree modeling, and the application of artificial intelligence to security analysis. Chapter 3 presents the theoretical and technical foundations necessary for understanding the system. It covers the principles of attack trees, the structure of the ISO/SAE 21434 standard, and core concepts behind large language models, retrieval techniques, and rule-based validation. Chapter 4 describes the overall methodology and system architecture. It explains the design and interaction of the key components, including retrieval, prompt generation, structured output formatting, and validation logic. Chapter 5 focuses on the technical implementation of the system, detailing the use of inference through quantized models, the setup of a semantic retriever, and the generation of rule-compliant attack trees. Chapter 6 presents the evaluation of the system based on a structured review of output quality, structural correctness, and the influence of retrieved context. Chapter 7 summarizes the results of the project, discusses its limitations, and outlines potential future improvements.

2 Related Works and Research Gap

This chapter reviews existing literature on the use of large language models and retrieval-augmented generation in the context of cybersecurity. It highlights a range of approaches to automated attack tree modeling, test generation, and security documentation, with a particular focus on the automotive domain and its specific requirements. The discussion emphasizes how AI-driven techniques are being used to support structured security tasks, and identifies where current methods fall short in terms of formal compliance, domain adaptation, or integration with threat modeling practices such as TARA. These observations lead into a clearly defined research gap that this thesis aims to address.

2.1 Related Works

The following works explore different uses of language models in cybersecurity, including attack tree generation, test case creation, and documentation support. Some focus on general-purpose systems, others on automotive contexts, and a few demonstrate the use of retrieval-based methods. Together, they provide useful reference points for positioning this thesis within the current state of research.

Dwight (2024) explores the use of generative AI for the creation of cyber attack trees based on official press releases from the U.S. Department of Justice [8]. The study compares outputs from three different Large Language Model (LLM)s: Microsoft Bing, Google Gemini, and ChatGPT 3.5, using a combination of quantitative analysis through analysis of variance (ANOVA) and qualitative analysis based on thematic evaluation. Results highlight notable differences in output complexity and structure, with ChatGPT producing more detailed and informative trees, while all models showed inconsistency in taxonomy and lacked refined logical constructs. While the study does not address structured compliance or domain-specific modeling, it provides valuable insight into the current limitations and potential of generative AI in attack tree generation. These findings support the motivation for further exploring rule-conformant generation of attack trees in more formalized domains such as automotive cybersecurity.

Khule (2023) presents Security Test Automation Framework (STAF), a framework for automating the generation of security test cases by leveraging existing attack trees as structured input [9]. Rather than generating attack trees itself, the system uses them as a semantic foundation to guide the creation of relevant security tests, aiming to systematically cover potential attack paths. Large language models are used to interpret the elements of the tree and generate test descriptions corresponding to attacker goals, steps, and system vulnerabilities. This approach demonstrates the potential of LLMs to work with structured security models, translating them into actionable outputs. While STAF operates in the reverse direction of this thesis by deriving content from attack trees instead of producing them, it supports the broader idea that LLMs can process and transform structured security knowledge effectively. This reinforces the viability of using LLMs for tasks like automated attack tree generation, provided the structure and semantics are well defined.

De Allende et al. propose a method for generating attack trees from natural language system descriptions using large language models in combination with the Common Attack Pattern Enumeration and Classification (CAPEC) database and custom heuristics [5]. Their implementation is embedded into the TTool environment and targets early-stage security design. The system uses feedback mechanisms to ensure structural validity and introduces metrics for evaluating the completeness and semantic quality of the generated trees. While their approach successfully demonstrates that LLMs can support structured attack tree generation, it differs from this thesis in several key aspects. Most notably, their work focuses on general-purpose systems and CAPEC-based threat modeling, whereas this thesis is situated in the automotive domain and follows the ISO/SAE 21434 threat and risk assessment methodology. Furthermore, their model operates as a closed system, without external knowledge retrieval. In contrast, this work explores RAG to incorporate domain-specific context such as historical attack trees, CVEs, and threat catalogs. These differences position this thesis as a more flexible and domain-adapted solution for generating rule-conformant attack trees in automotive cybersecurity.

Feng et al. present Mixture of RAGs Security Expert (MoRSE), a modular Retrieval-Augmented Generation framework designed to assist security engineers in producing high-quality security documentation across a variety of tasks [4]. The architecture integrates retrieval, content formatting, classification, summarization, and generation stages, with flexibility to combine different tools and language models depending on the use case. Their results, evaluated through expert feedback and automated scoring, demonstrate that RAG-based pipelines can improve the quality, consistency, and relevance of security-focused outputs. While MoRSE focuses on generating unstructured reports and documentation, its modular approach and evaluation results provide valuable insight for other domains. This thesis builds on the same foundational idea that retrieval can ground language model output in relevant prior knowledge. By applying this principle to the structured generation of attack trees in the automotive domain, the work here extends the scope of RAG-based automation toward formal modeling use cases within the TARA process.

Chlup et al. introduce THREATGET, a tool-based framework for the automated generation and analysis of attack trees within the automotive domain [2]. The system is integrated into model-based design environments and leverages predefined rule sets and threat libraries to construct attack trees from SysML-based system architectures. Its primary aim is to support the TARA process in line with ISO/SAE 21434 by improving consistency, traceability, and efficiency in early development phases. Although THREATGET does not use machine learning or language models, it clearly demonstrates the need for automation in attack tree modeling and validates the role of structured threat modeling in automotive cybersecurity. This thesis shares that motivation but approaches the problem from a different angle: instead of relying on structured system models, it investigates whether large language models combined with retrieval can generate meaningful attack trees directly from natural language threat descriptions. THREATGET thus reinforces the relevance of attack tree automation while also highlighting a key distinction — this thesis explores whether LLM-based generation can offer a more flexible, text-driven alternative when structured design models are not available.

Paper Title	**Authors**	**Key Contribution**
Building Cyber Attack Trees with the Help of My LLM?	Joshua Dwight	Compares outputs of Bing, Gemini, and ChatGPT for attack tree generation from DOJ press releases. Highlights inconsistencies and lack of structure, supporting the need for rule-based generation in formal domains.
STAF: Leveraging LLMs for Automated Attack Tree-Based Security Test Generation	Tanmay Khule	Uses existing attack trees as structured input to generate security tests. Demonstrates that LLMs can translate structured threat models into actionable test descriptions.
Automated Attack Tree Generation Using Artificial Intelligence and Natural Language Processing	Alan Birchler De Allende, Bastien Sultan, Ludovic Apvrille	Generates attack trees from natural language using heuristics and CAPEC data. Focuses on general-purpose systems with embedded feedback but lacks retrieval augmentation.
MoRSE: Bridging the Gap in Cybersecurity Expertise with Retrieval Augmented Generation	Marco Simoni, Andrea Saracino, Vinod P, Mauro Conti	Modular RAG framework for security documentation. Offers reusable architecture and evaluation strategy for retrieval-grounded generation.
THREATGET: Towards Automated Attack Tree Analysis for Automotive Cybersecurity	Sebastian Chlup, Korbinian Christl, Christoph Schmittner, Abdelkader Magdy Shaaban, Stefan Schauer, Martin Latzenhofer	Rule-based attack tree generation tool for SysML system models. Highlights value of automation in TARA but does not use LLMs.

Table 2.1: Overview of selected related works and their contributions [2, 4, 5, 8, 9]

2.2 Identified Research Gap

Although several studies have explored the use of large language models in cybersecurity-related tasks, the automated generation of structured, rule-compliant attack trees in the automotive domain remains underexplored. Existing approaches either focus on general-purpose use cases or rely on simplified structures that do not reflect the complexity required by formal methodologies such as ISO/SAE 21434.

Efforts to apply language models to security modeling have primarily explored informal or exploratory use cases. These approaches often prioritize readability over structural precision, leaving out critical aspects required for formal attack tree construction. Others, like STAF, showcase the potential of language models to work with attack trees by generating test cases from them, but do not address the challenge of generating the trees themselves. MoRSE, while focused on generating security documentation, provides valuable evidence that retrieval-based pipelines can help language models produce more contextually grounded results. However, its outputs are not designed for structured security modeling and do not follow the rules or conventions of formal threat analysis.

Approaches that attempt structured generation, such as the work by De Allende et al., do not incorporate external knowledge through retrieval and are not tailored to the automotive context. Similarly, rule-based tools like THREATGET operate without Artificial Intelligence (AI) and depend on formal system models, limiting flexibility when threat scenarios originate from natural language descriptions.

What is missing is a method that combines domain-aware retrieval with language model generation to produce attack trees that are both contextually grounded and structurally valid. This thesis addresses that gap by exploring a retrieval-based generation pipeline for attack tree modeling within the context of automotive TARA.

3 Background

This chapter lays the foundation for understanding the core concepts and methodologies that underpin automated attack tree generation for TARA in automotive systems. It explores modern vehicle cybersecurity challenges, the structured threat modeling process, the principles of retrieval-augmented generation, and the tools and standards that shape industrial implementation. These elements provide essential context for the approach developed later in this thesis.

3.1 Automotive Cybersecurity

Automotive cybersecurity focuses on protecting the safety, confidentiality, integrity, and availability of systems within modern vehicles. As software and connectivity increasingly define vehicle functions, cybersecurity has become essential to ensure not only data protection but also the operational safety of automotive systems. Unlike traditional IT environments, cyber attacks in vehicles may affect physical systems directly, which includes braking, steering, or acceleration.

Modern vehicles integrate up to 150 ECUs and run tens of millions of lines of code. This complexity, combined with constant communication over wireless and wired interfaces, significantly increases the vehicle's attack surface [1]. Connected cars continuously interact with cloud services, external devices, and infrastructure, exposing a wide range of potential attack vectors. These include infotainment systems, telematics units, USB ports, wireless communication, and over-the-air update mechanisms [10].

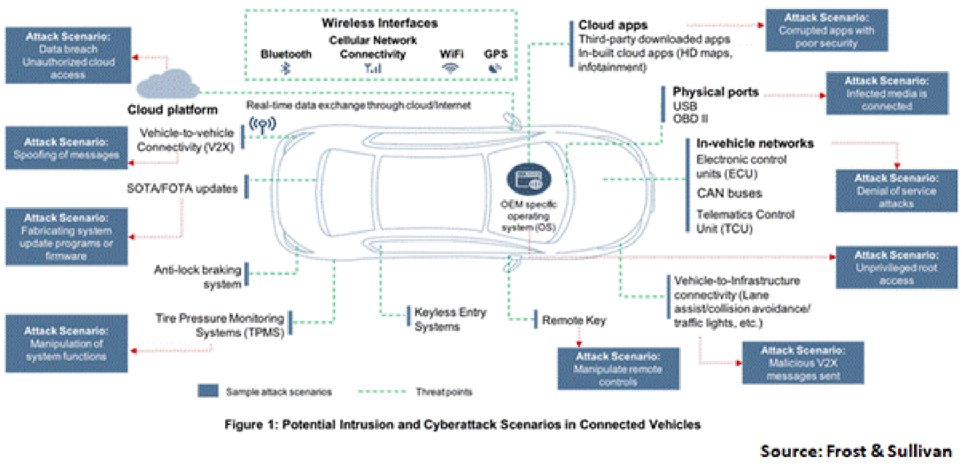
Attack Scenario: Data breach Unauthorized cloud access
Wireless Interfaces
Bluetooth
Cellular Network Connectivity
WiFi
GPS
Cloud apps
Third-party downloaded apps
In-built cloud apps (HD maps, infotainment)
Attack Scenario: Corrupted apps with poor security
Cloud platform
Real-time data exchange through cloud/Internet
Physical ports
USB
OBD II
Attack Scenario: Infected media is connected
Attack Scenario: Spoofing of messages
Vehicle-to-vehicle Connectivity (V2X)
In-vehicle networks
Electronic control units (ECU)
CAN buses
Telematics Control Unit (TCU)
SOTA/FOTA updates
OEM specific operating system (OS)
Attack Scenario: Denial of service attacks
Attack Scenario: Fabricating system update programs or firmware
Anti-lock braking system
Attack Scenario: Unprivileged root access
Vehicle-to-Infrastructure connectivity (Lane assist/collision avoidance/ traffic lights, etc.)
Tire Pressure Monitoring Systems (TPMS)
Keyless Entry Systems
Remote Key
Attack Scenario: Manipulation of system functions
Attack Scenario: Manipulate remote controls
Attack Scenario: Malicious V2X messages sent
Sample attack scenarios
Threat points
Figure 1: Potential Intrusion and Cyberattack Scenarios in Connected Vehicles
Source: Frost & Sullivan

Fig. 3.1: (Placeholder for now) Potential Intrusion and Cyberattack Scenarios in Connected Vehicles ([10])

Figure 3.1 illustrates various cyberattack entry points and threat scenarios in connected vehicles. These range from spoofing messages over Vehicle-to-everything (V2X) channels and injecting malicious firmware into ECUs, to exploiting remote key access or tampering with diagnostic tools. Attacks like these are no longer theoretical; they represent real-world risks that demand systematic threat modeling and rigorous assessment during the development lifecycle.

To mitigate these threats, automotive cybersecurity frameworks, such as ISO/SAE 21434, emphasize structured TARA. Within this process, the modeling of attack paths plays a central role in understanding how a system can be compromised, enabling engineers to apply protective measures proactively.

3.2 ISO/SAE 21434

ISO/SAE 21434, titled "Road Vehicles — Cybersecurity Engineering", is an internationally recognized standard that defines a framework for managing cybersecurity risks in the automotive domain [3]. The standard was developed in response to the growing digital complexity of modern vehicles and the resulting need for structured, lifecycle-spanning cybersecurity processes.

As ECUs, software-defined components, and connected interfaces become integral to vehicle operation, the potential for cyber threats affecting both safety and data security has increased significantly. ISO/SAE 21434 provides guidelines to identify, assess, and manage these risks in a consistent and traceable way. It applies to all road vehicles, including cars, trucks, buses, and motorcycles, and is intended for use by manufacturers, suppliers, and engineering service providers involved in vehicle development.

The standard structures cybersecurity activities along the entire product lifecycle, covering phases from concept and development to production, operation, maintenance, and decommissioning. It includes requirements for risk management, organizational processes, technical development, and supporting activities. The standard is aligned with international regulations such as UNECE WP.29 and supports compliance with Cybersecurity Management System (CSMS) and Software Update Management System (SUMS) requirements. Its integration into development workflows ensures that cybersecurity is treated not as a one-time consideration, but as a continuous activity throughout the vehicle's lifecycle.

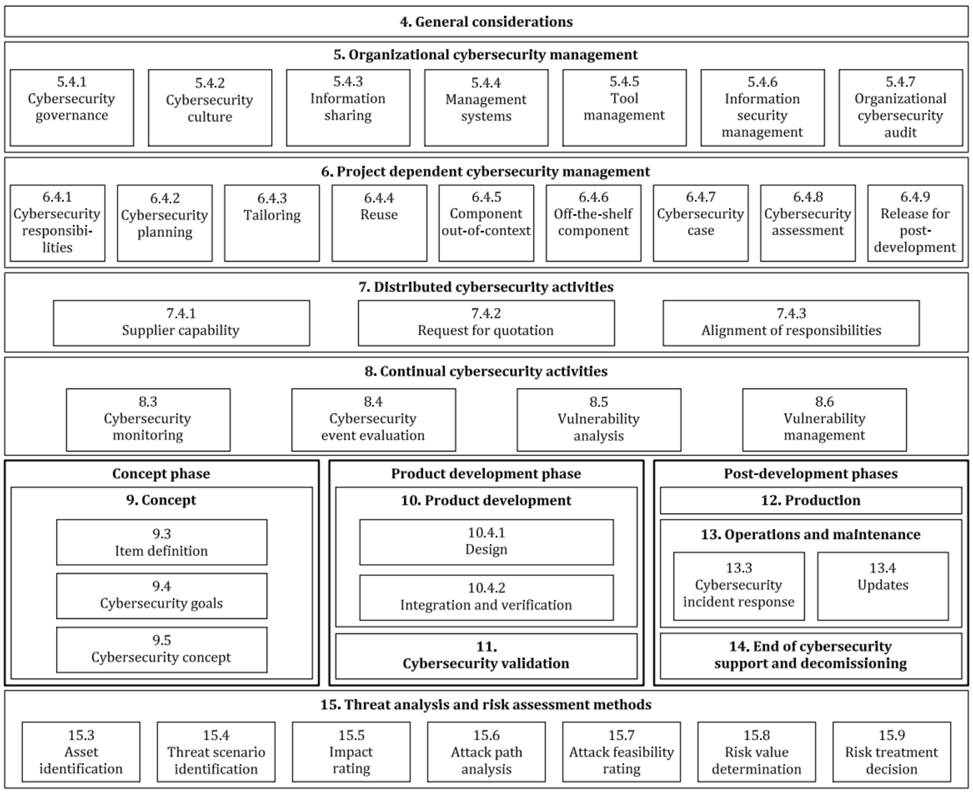
4. General considerations
5. Organizational cybersecurity management
5.4.1 Cybersecurity governance
5.4.2 Cybersecurity culture
5.4.3 Information sharing
5.4.4 Management systems
5.4.5 Tool management
5.4.6 Information security management
5.4.7 Organizational cybersecurity audit
6. Project dependent cybersecurity management
6.4.1 Cybersecurity responsibi-lities
6.4.2 Cybersecurity planning
6.4.3 Tailoring
6.4.4 Reuse
6.4.5 Component out-of-context
6.4.6 Off-the-shelf component
6.4.7 Cybersecurity case
6.4.8 Cybersecurity assessment
6.4.9 Release for post-development
7. Distributed cybersecurity activities
7.4.1 Supplier capability
7.4.2 Request for quotation
7.4.3 Alignment of responsibilities
8. Continual cybersecurity activities
8.3 Cybersecurity monitoring
8.4 Cybersecurity event evaluation
8.5 Vulnerability analysis
8.6 Vulnerability management
Concept phase
9. Concept
9.3 Item definition
9.4 Cybersecurity goals
9.5 Cybersecurity concept
Product development phase
10. Product development
10.4.1 Design
10.4.2 Integration and verification
11. Cybersecurity validation
Post-development phases
12. Production
13. Operations and maintenance
13.3 Cybersecurity incident response
13.4 Updates
14. End of cybersecurity support and decomissioning
15. Threat analysis and risk assessment methods
15.3 Asset identification
15.4 Threat scenario identification
15.5 Impact rating
15.6 Attack path analysis
15.7 Attack feasibility rating
15.8 Risk value determination
15.9 Risk treatment decision

Fig. 3.2: Structure of the ISO/SAE 21434 standard [3]

Figure 3.2 illustrates how the standard is structured across different phases, covering organizational policies, project-level risk management, and post-development activities. This structure helps ensure that cybersecurity considerations are addressed consistently throughout the entire product lifecycle.

3.2.1 Threat Analysis and Risk Assessment (TARA)

Threat Analysis and Risk Assessment (TARA) is a core process described in ISO/SAE 21434 that supports the identification, evaluation, and treatment of cybersecurity risks across the vehicle lifecycle. The process reflects the secure-by-design principle and is initiated early in the development phases to ensure that potential attack vectors are systematically identified and mitigated.

TARA consists of several logically connected steps that form a structured methodology. These steps are grouped into six main phases: *Context Establishment*, *Damages*, *Threats*, *Attacks*, *Risks*, and *Security Goals*. The process is shown in Figure 3.3, which visualizes how the outputs of each phase feed into the next.

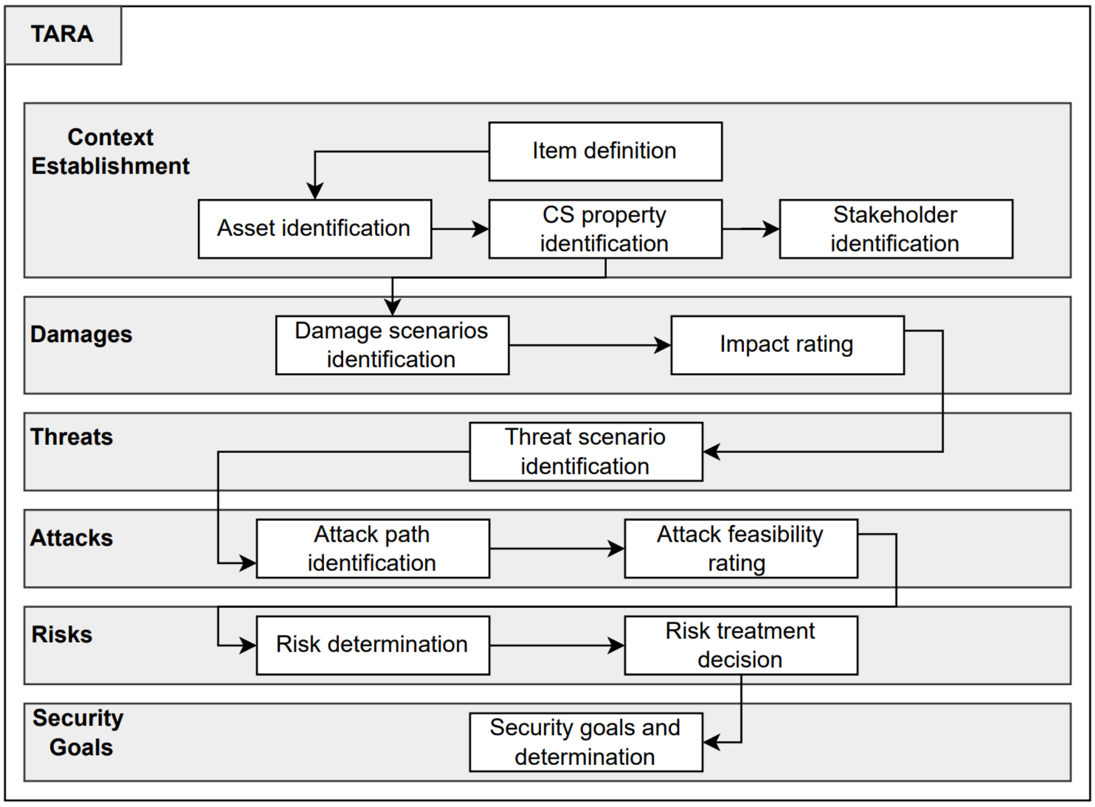
TARA
Context Establishment
Item definition
Asset identification
CS property identification
Stakeholder identification
Damages
Damage scenarios identification
Impact rating
Threats
Threat scenario identification
Attacks
Attack path identification
Attack feasibility rating
Risks
Risk determination
Risk treatment decision
Security Goals
Security goals and determination

Fig. 3.3: TARA process structure based on ISO/SAE 21434 [11]

In the first phase, **Context Establishment**, the item under analysis is defined, including its interfaces and operational scope. This is followed by the identification of relevant assets and the cybersecurity properties that need protection—such as confidentiality, integrity, and availability. Stakeholders and their security expectations are also documented.

The second phase, **Damages**, involves identifying damage scenarios that describe what could happen if cybersecurity objectives are violated. These scenarios are then rated based on their potential impact on safety, privacy, financial value, and operations.

In the **Threats** phase, threat scenarios are derived from known or assumed vulnerabilities related to the identified assets. As described by Vielberth et al. [12], this step is crucial for ensuring that the analysis remains focused on realistic adversarial behavior and system-specific exposure.

The **Attacks** phase models how threats can be carried out. Attack paths are constructed using structured methods such as attack trees, which break down complex intrusions into smaller, traceable steps. These paths are evaluated for feasibility based on attacker capabilities, access conditions, and technical barriers.

Next, in the **Risks** phase, both the impact and feasibility assessments are combined to determine the severity of each threat scenario. Based on this, appropriate treatment decisions are made—such as implementing countermeasures, reducing exposure, or accepting residual risks.

Finally, the **Security Goals** phase derives high-level objectives from the treated risks. These goals define what the system must achieve from a cybersecurity standpoint and guide downstream implementation decisions.

Together, these steps form a complete lifecycle for cybersecurity risk assessment. The TARA process not only enables early threat identification but also ensures that security decisions are traceable and consistent throughout development.

3.3 Attack Trees for Threat Modeling

Attack trees are a widely used formalism for modeling how an adversary might compromise a system. Originally introduced by Bruce Schneier [13] in the late 1990s, attack trees provide a graphical and hierarchical way to describe security threats, starting from a general attacker objective and decomposing it into sub-goals and concrete attack steps. Each path through the tree represents a possible sequence of actions an attacker could follow to achieve a particular malicious outcome.

At the top of the tree is the root node, which represents the attacker's high-level goal, such as "Disable Braking System" or "Extract Confidential Firmware." This goal is then recursively broken down into intermediate nodes, which denote attacker sub-goals, and finally into leaf nodes, which describe atomic actions or techniques that can be executed directly (e.g., "Inject CAN Message" or "Exploit USB Port"). Logical relationships between these nodes are expressed using AND and OR gates. An AND gate signifies that all child nodes must be fulfilled to satisfy the parent, while an OR gate indicates that any one of the children is sufficient.

The strength of attack trees lies in their combination of structure, visual clarity, and evaluability. By modeling attack paths systematically, they allow security analysts to trace how vulnerabilities propagate, compare different attack vectors, and assess the conditions required for each step. This structured representation is particularly well suited to formal threat analysis processes like TARA as defined in ISO/SAE 21434, where traceability, rating propagation, and goal decomposition are essential.

Modern approaches increasingly rely on attack trees to encode not just potential attack sequences but also the effort, resources, and expertise required to carry them out. These attributes form the basis for various risk metrics, enabling the prioritization of threats and supporting defensive decisions.

This thesis builds on these foundations by exploring whether large language models can be used to automatically generate attack trees that follow the same hierarchical structure and logic as those produced by human experts.

3.3.1 Structural Concepts and Formal Semantics

Attack trees are structured as directed, acyclic graphs in which nodes represent attacker goals or actions and edges represent logical decomposition. The formal semantics of attack trees allow for both visual clarity and algorithmic evaluation, making them a powerful tool for modeling adversarial behavior in cybersecurity contexts [13, 14].

Each node in an attack tree can be categorized into one of three levels:

- **Root node:** Represents the ultimate goal of the attacker (e.g., "Disable Vehicle Function").
- **Intermediate nodes:** Capture sub-goals or stages toward achieving the main goal (e.g., "Gain Control Over ECU").
- **Leaf nodes:** Describe atomic attack actions or concrete technical steps (e.g., "Exploit Diagnostic Interface").

Two gate types are typically used to define the logical relationship between parent and child nodes:

- **AND gate:** All child nodes must be successfully executed to satisfy the parent node.
- **OR gate:** At least one child node is sufficient to satisfy the parent node.

These gates define a deterministic evaluation logic, which supports consistent risk propagation across the tree. For example, under an OR gate, the most feasible child node (e.g., lowest attack complexity or effort) determines the feasibility of the parent. For AND gates, the parent inherits the most difficult (least feasible) attribute among its children [15]. This enables a bottom-up feasibility estimation that aligns with structured modeling practices. A detailed rating logic based on attacker effort for the feasability is introduced in chapter 3.3.2, using a multidimensional attack-potential-based approach that incorporates attributes such as required time, expertise, and equipment.

Modern approaches to attack tree generation, particularly in regulated domains like automotive cybersecurity, require adherence to strict structural rules and formal semantics. As described by Konsta et al. [14], formalization is essential for automation and integration with security frameworks and tools.

In this thesis, the generated attack trees are required to follow these conventions: node types, gate logic, and rating fields must be well-defined and consistently propagated.

3.3.2 Attributes and Evaluation in Attack Trees

To quantify risks in attack trees, each node must be evaluated not only in terms of its function within the structure, but also based on well-defined attributes that describe the feasibility of a successful attack. This facilitates a bottom-up evaluation, where leaf nodes represent concrete actions or attack steps, and parent nodes reflect the feasibility of higher-level attacker goals, based on the logic of their child components.

A widely used approach to estimate attack feasibility is the **attack potential-based model** [16]. It is derived from ISO/IEC 18045:2008 and adapted to the automotive domain by incorporating context-specific factors. Each leaf node is assessed along five key dimensions:

Attribute	**Possible Values**
Elapsed Time	$\leq$ 1 day, $\leq$ 1 week, $\leq$ 1 month, $\leq$ 6 months, $>$ 6 months
Expertise	Layman, Proficient, Expert, Multiple Experts
Knowledge of Item	Public, Restricted, Confidential, Strictly Confidential
Window of Opportunity	Unlimited, Easy, Moderate, Difficult
Equipment	Standard, Specialized, Bespoke, Multiple Bespoke

Table 3.1: Feasibility attributes for attack potential-based evaluation

Each attribute is mapped to a numerical value from 0 (least demanding) to 3 (most demanding), and the total score determines the attack feasibility rating. This value is then classified into qualitative categories such as *High*, *Medium*, *Low* or *Very Low* feasibility, depending on thresholds defined by the scoring framework [16].

Bottom-Up Propagation via Logical Gates

The risk associated with a parent node is propagated from its children based on the logical structure of the tree. For an `OR` gate, the child node with the highest feasibility (i.e., the lowest overall attack potential score) determines the feasibility rating of the parent. In contrast, an `AND` gate propagates each attribute individually: the parent inherits the most difficult value (e.g., longest elapsed time, highest required expertise) per attribute across all child nodes. This means that the parent node may accumulate a higher overall attack potential score than any individual children, since it represents the compounded effort needed to execute all required attack steps. This bottom-up propagation ensures that the final feasibility ratings remain logically consistent and structurally meaningful throughout the attack tree.

3.3.3 Transformation and Normal Forms

Attack trees are not uniquely defined in terms of their structure. Different tree topologies can encode logically equivalent attack scenarios, particularly when considering the use of logical gates such as `AND` and `OR`. For example, flattening nested `OR` gates or reordering children under commutative gates does not affect the semantics of the attack tree. This property allows for transformation and normalization operations that preserve meaning while altering structure [15].

A key concept is **normal form**, a standardized representation of an attack tree where structural redundancies are removed. Reduction rules include:

- **Gate flattening:** Replace nested gates of the same type with a single-level gate.
- **Single-child removal:** Eliminate unnecessary intermediate nodes with only one child.
- **Idempotency:** Remove duplicated subtrees under the same parent.

These transformations are supported by formal equivalence relations. Let T_1 and T_2 be two trees, then:

$$T_1 \equiv T_2 \iff \forall \text{ valuation } v : T_1[v] = T_2[v]$$

This formula states that two trees are semantically equivalent if their evaluation yields the same result under all possible attacker models or feasibility interpretations.

In the context of LLM-based generation, this flexibility is essential. Generated trees may differ in structure but still encode the same attack logic. Recognizing and reducing such variations can improve downstream processing, reduce redundancy, and simplify validation.

3.4 Large Language Models

Large language models (LLMs) are a class of artificial intelligence systems trained to process and generate human-like language. These models are built upon neural network architectures and trained on massive corpora of textual data, allowing them to learn statistical patterns, semantic relationships, and syntactic structures across languages. By predicting the next token in a sequence given a context, LLMs can perform a wide variety of natural language tasks, including translation, summarization, question answering, and code generation [17].

Unlike earlier rule-based systems or simpler statistical models, LLMs do not require task-specific engineering. Instead, their capabilities emerge from the scale of their training data and model parameters. Key to their success is the transformer architecture, which introduced self-attention mechanisms to model long-range dependencies in text more effectively than recurrent or convolutional methods [18]. Since then, transformer-based LLMs have become the de facto standard in NLP and AI applications.

3.4.1 Transformer Architecture

The transformer is a deep learning architecture introduced by Vaswani et al. in 2017. It relies entirely on self-attention mechanisms, discarding recurrence and convolutions altogether, and has been foundational in the design of LLMs like GPT, BERT, and LLaMA [18].

The core idea of the transformer is to compute attention scores that indicate how strongly each word in a sequence should attend to every other word. These scores are computed in parallel across multiple attention heads, each capturing different aspects of the sentence semantics. The architecture consists of encoder and decoder blocks stacked on top of each other, where each block includes multi-head self-attention, feed-forward layers, residual connections, and layer normalization.

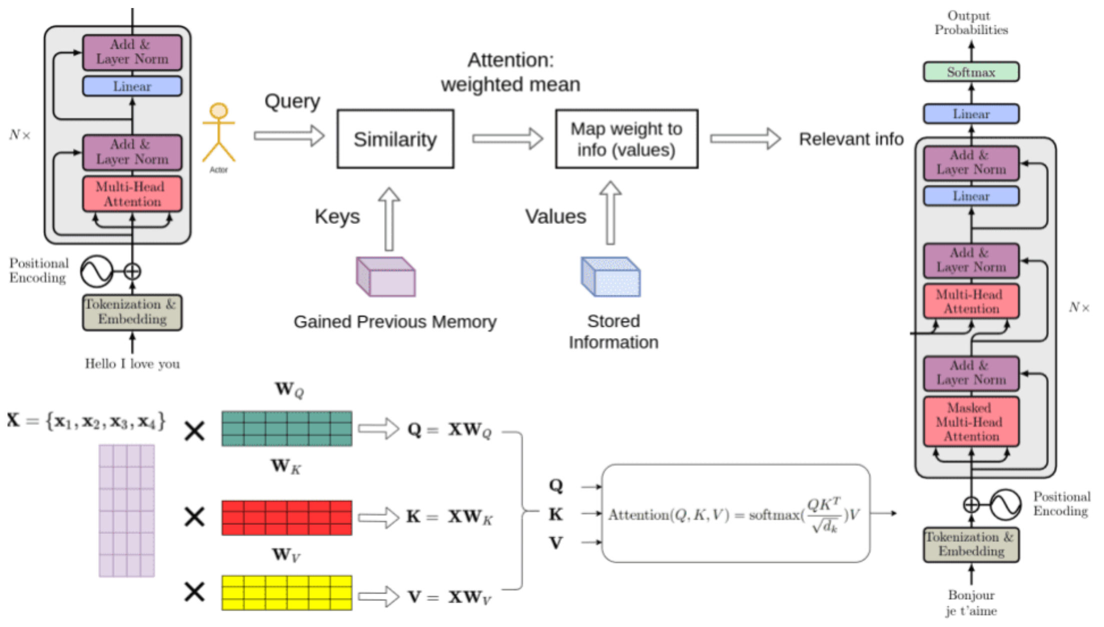

Fig. 3.4: Structure of a Transformer Encoder Block [19]

Figure 3.4 illustrates the inner workings of a Transformer encoder block, a core component of large language models. The processing pipeline follows a sequence of transformations designed to extract and model contextual meaning from input tokens.

The first step involves tokenization, where the input sentence is segmented and converted into numerical representations, which are called tokens. These tokens are embedded into high-dimensional vector spaces that capture semantic properties of words [17]. Since transformers lack an inherent notion of word order, positional encodings are added to each word embedding to inject information about token positions within the sequence [18].

The resulting vectors are passed into the encoder stack, which is composed of multiple identical layers. Each encoder layer includes:

- **Multi-Head Self-Attention:** This mechanism allows the model to assess relationships between all pairs of words in a sentence. Each attention head focuses on different aspects of the context (e.g., syntactic vs. semantic relations), which are then concatenated and linearly transformed [18].

- **Add & Normalize:** The output of the attention module is combined with the input via a residual connection and normalized using layer normalization. This stabilizes training and helps prevent vanishing gradients [20].
- **Feed-Forward Network (MLP):** A fully connected two-layer neural network is applied independently to each position in the sequence. This component allows the model to apply nonlinear transformations to the attention output.
- **Second Add & Normalize:** The MLP output is again passed through a residual connection and normalization layer, completing one encoder block.

These encoder blocks are stacked multiple times (e.g., six in the original model), allowing the model to refine its representation of sentence structure across multiple abstraction layers. This iterative architecture enables the Transformer to capture both local and global dependencies, making it effective for a broad range of natural language processing tasks [17, 18].

The use of attention allows transformers to model complex dependencies regardless of distance, making them ideal for tasks involving long or structured text. Additionally, their parallelizable architecture leads to faster training and inference compared to RNN-based models.

The power of transformer models lies in their flexibility and scalability: as model size and training data grow, their ability to generalize across language tasks improves significantly [17]. This architecture remains the backbone of all major LLMs today.

3.4.2 Model Quantization Techniques and Local Inference

As large language models (LLMs) grow in size, deploying them efficiently becomes increasingly difficult. Two important approaches to address this challenge are quantization and local inference.

Quantization reduces the storage size and computational cost of LLMs by representing model parameters with lower-bit precision. Instead of storing each weight as a 32-bit floating point number, quantized models use formats such as 8-bit integers (INT8), or even lower (e.g., 4-bit, 2-bit), resulting in significantly smaller models. For example, a 4-bit quantized model requires only 12.5% of the memory of its full-precision equivalent. This compression also allows for faster inference due to better hardware utilization and reduced memory bandwidth [21].

However, these gains come with trade-offs. Lower-precision formats may lead to a drop in model accuracy or increase numerical instability, especially in sensitive layers. To mitigate this, advanced techniques such as quantization-aware training or mixed-precision strategies are used to preserve performance while minimizing resource demands. The balance between efficiency and model fidelity must be carefully managed.

Local inference builds on this by enabling these smaller, quantized models to be executed on systems with lower performance. This approach enhances data privacy, eliminates network latency, and reduces deployment costs. Projects like Ollama and llama.cpp make this possible by combining quantized weights with optimized inference backends, allowing large models to run on commodity GPUs or even CPUs [22].

Together, quantization and local inference make it feasible to use LLMs in constrained or privacy-sensitive environments while maintaining acceptable performance levels.

3.4.3 Language Model Behavior

Large Language Models (LLMs) function as probabilistic sequence predictors. At their core, they estimate the likelihood of the next token in a sequence given all preceding tokens. This behavior stems from their training objective, which is to minimize the cross entropy loss between the predicted token distribution and the actual next token in a large corpus of text [17]. As a result, LLMs do not retrieve facts from a knowledge base but generate responses based on statistical patterns found in the training data. This results in both impressive generative capabilities and certain fundamental limitations.

One such limitation is hallucination, which refers to the generation of outputs that appear fluent and contextually appropriate but are factually incorrect or not supported by the input [23, 24]. Hallucinations can occur when the model confidently predicts tokens that are unlikely or do not conform to the learned distribution. These issues become more pronounced when models exceed their context window or are prompted with ambiguous or underspecified inputs. One such limitation is hallucination, which refers to the generation of outputs that appear fluent and contextually appropriate but are factually incorrect or not supported by the input [23, 24]. Hallucinations can occur when the model confidently predicts tokens that are unlikely or do not conform to the learned distribution. These issues become more pronounced when models exceed their context window or are prompted with ambiguous or underspecified inputs.

In addition, LLMs are sensitive to the exact wording of inputs. Even small variations in phrasing can lead to significantly different outputs, which illustrates the brittle and non-deterministic nature of their behavior. This variability arises from their reliance on learned associations and sampling strategies instead of deterministic reasoning processes [17].

Recent methods aim to detect hallucinations by analyzing features such as token probability variance and vocabulary-level confidence scores. For instance, Quevedo et al. introduce a supervised learning approach that uses simple statistical features—specifically, the minimum and average token probabilities, along with the deviation from the model's top predictions—to identify hallucinated content across different LLMs. Their study shows that these indicators can help flag unreliable outputs, even when the evaluating model differs from the one that generated the response. As stated in their work, "We show that even simple statistics over model probabilities can serve as strong indicators of hallucinations" [23].

These behavioral characteristics are especially important when applying LLMs to structured or safety-critical tasks, such as the generation of threat models. Understanding and addressing these behaviors is essential for ensuring consistency, reliability, and trustworthiness in the outputs of such systems.

3.4.4 Prompt Engineering and Model Fine-Tuning

Large Language Models (LLMs) such as LLaMA and GPT variants offer a broad spectrum of capabilities that can be adapted to specific tasks through two primary methods: prompt engineering and model fine-tuning. Both approaches aim to align model outputs with domain-specific goals, yet they differ fundamentally in how this alignment is achieved.

Prompt Engineering

Prompt engineering involves designing input prompts that guide the model to produce more accurate, relevant, or structured outputs without modifying the model's internal weights. This technique leverages the model's existing knowledge by shaping the context in which it generates responses. Strategies include few-shot prompting, zero-shot prompting, and chain-of-thought prompting, each intended to elicit specific reasoning patterns or output structures [25].

In Retrieval-Augmented Generation (RAG), prompt engineering plays a crucial role by embedding retrieved context into the prompt template. This fusion of external knowledge and carefully structured instruction increases the likelihood that the model produces accurate and domain-relevant output while mitigating hallucinations. Prompt engineering is especially valuable in settings where access to fine-tuning is limited due to resource constraints, privacy concerns, or the need for rapid iteration.

According to Marvin et al. [25], prompt engineering has evolved into a programming paradigm in its own right. The authors emphasize the importance of defining clear goals, understanding model limitations, providing domain-specific context, and iteratively refining prompts to guide the model's reasoning. Advanced prompting techniques, such as dynamic prompting, contextual prompting, and programmatic instruction generation, enable improved control over model outputs. These methods enhance few-shot learning performance, support domain adaptation, and expand the applicability of LLMs across diverse NLP tasks, including question answering, text generation, reasoning, and dialogue modeling.

Model Fine-Tuning

While prompt engineering influences model behavior through inputs, fine-tuning involves modifying the model's internal parameters by training on new, task-specific data. Fine-tuning can be performed in different forms depending on the computational and data requirements:

- **Full fine-tuning** updates all model weights, requiring substantial compute and memory resources. This approach is suitable when extensive adaptation to a new domain is necessary.
- **Parameter-Efficient Fine-Tuning (PEFT)** methods, such as prefix tuning, adapter modules, and Low-Rank Adaptation (LoRA), selectively adjust a small subset of parameters or inject task-specific modules into the network, drastically reducing the training footprint.
- **LoRA (Low-Rank Adaptation)** in particular has emerged as a scalable method for adapting large models using low-rank matrix decompositions applied to specific weight matrices. It offers a favorable trade-off between performance and efficiency, and supports multi-domain adaptation without duplicating full model checkpoints.

According to Mohammed and Kora [26], PEFT approaches like LoRA not only reduce the cost and complexity of deploying fine-tuned models, but also enable modularity and reuse across different tasks. Their survey highlights how LoRA maintains performance comparable to full fine-tuning in many downstream applications, including machine translation, summarization, and domain-specific text generation.

In safety-critical or resource-constrained environments, fine-tuning is often combined with prompt engineering to balance flexibility, control, and computational efficiency. This hybrid approach is especially useful in domains such as automotive cybersecurity, where strict formatting, structural rules, and domain fidelity are essential for practical adoption.

3.5 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) combines large language models (LLMs) with external knowledge retrieval to enhance factual accuracy and reduce hallucinations. Instead of relying only on static parametric memory encoded in the model's weights, RAG pipelines query an external knowledge base, such as a vector database, to dynamically provide relevant information during inference. This architecture makes it possible to ground answers in verifiable sources and to update the model's effective knowledge without retraining [27, 28].

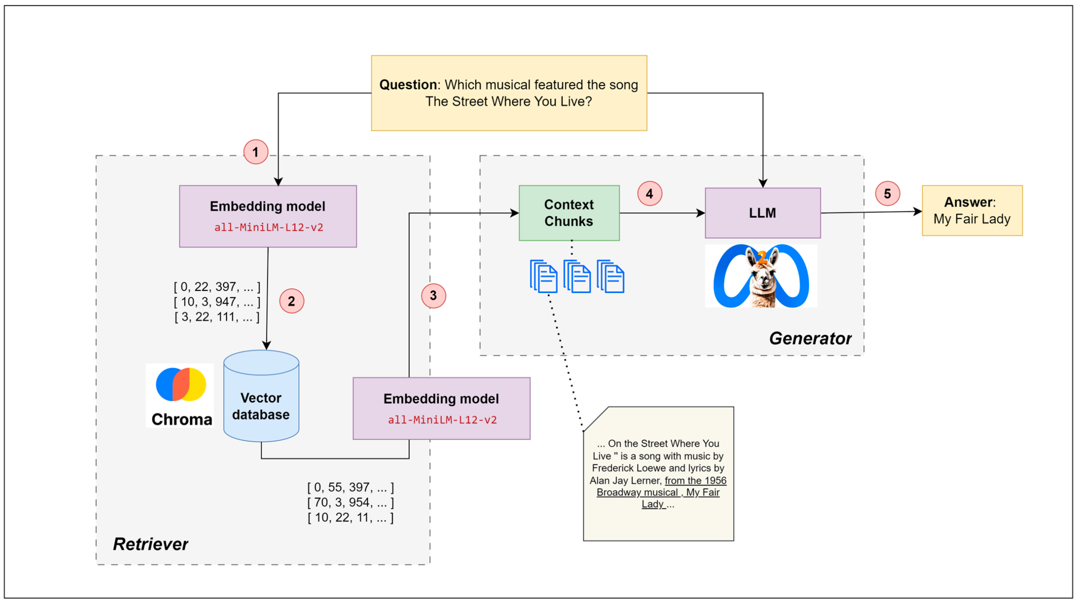

Fig. 3.5: RAG pipeline overview with retriever and generator [27]

The architecture consists of two major components: a **retriever** and a **generator**. The retriever embeds the input query into a dense vector space and compares it against pre-computed document embeddings in a vector index. It selects the top-k most relevant documents, where k denotes how many candidate passages are returned. A small k may miss important context, while a larger k risks introducing irrelevant or noisy passages. These retrieved passages are concatenated with the query and passed to the generator, typically a transformer-based LLM, which produces the final response.

3.5.1 Semantic Embeddings

Semantic embeddings map natural language into high-dimensional numerical vectors that capture meaning rather than surface-level word overlap. For a given query q and document d, their embeddings are obtained through neural encoders:

$$\mathbf{q} = \mathrm{Encoder}_q(q), \quad \mathbf{d} = \mathrm{Encoder}_d(d).$$

Here, both encoders are often transformer-based models such as BERT or MiniLM. The result of this operation is that semantically similar texts are represented by vectors that lie close to each other in the embedding space. This property enables the retriever to find documents relevant to the query even when they share little or no lexical overlap.

3.5.2 Cosine Similarity

To measure how closely a query vector $\mathbf{q}$ matches a document vector $\mathbf{d}$, a similarity function is applied. One of the most common measures is cosine similarity:

$$\mathrm{sim}(\mathbf{q}, \mathbf{d}) = \frac{\mathbf{q} \cdot \mathbf{d}}{\|\mathbf{q}\| \|\mathbf{d}\|}.$$

The result is a value between -1 and 1, where 1 means that the query and document are identical in direction (perfect semantic alignment), 0 means no relation, and -1 indicates complete opposition. In practice, higher values indicate that the document is more relevant to the query. The retriever then ranks documents by their similarity score and selects the top-k results.

3.5.3 Dense Retrieval with DPR and FAISS

Dense retrieval is a core component of modern Retrieval-Augmented Generation (RAG) systems. Unlike sparse methods such as BM25, dense retrievers use semantic embeddings to represent queries and documents in a shared vector space [11]. A widely used approach is **Dense Passage Retrieval (DPR)**, which applies a dual-encoder model to embed queries and passages independently. Their similarity is calculated using the dot product. The training process uses contrastive loss with hard negatives, which improves the model's ability to rank relevant documents [29].

The resulting document embeddings are stored in a vector index using Facebook AI Similarity Search (FAISS). During inference, the query is embedded and compared to the index using Approximate Nearest Neighbor (ANN) search. The top-k most relevant documents are retrieved based on similarity scores. Choosing a value for k involves a trade-off. A small k can miss relevant context, while a large k may introduce unrelated or distracting information [28].

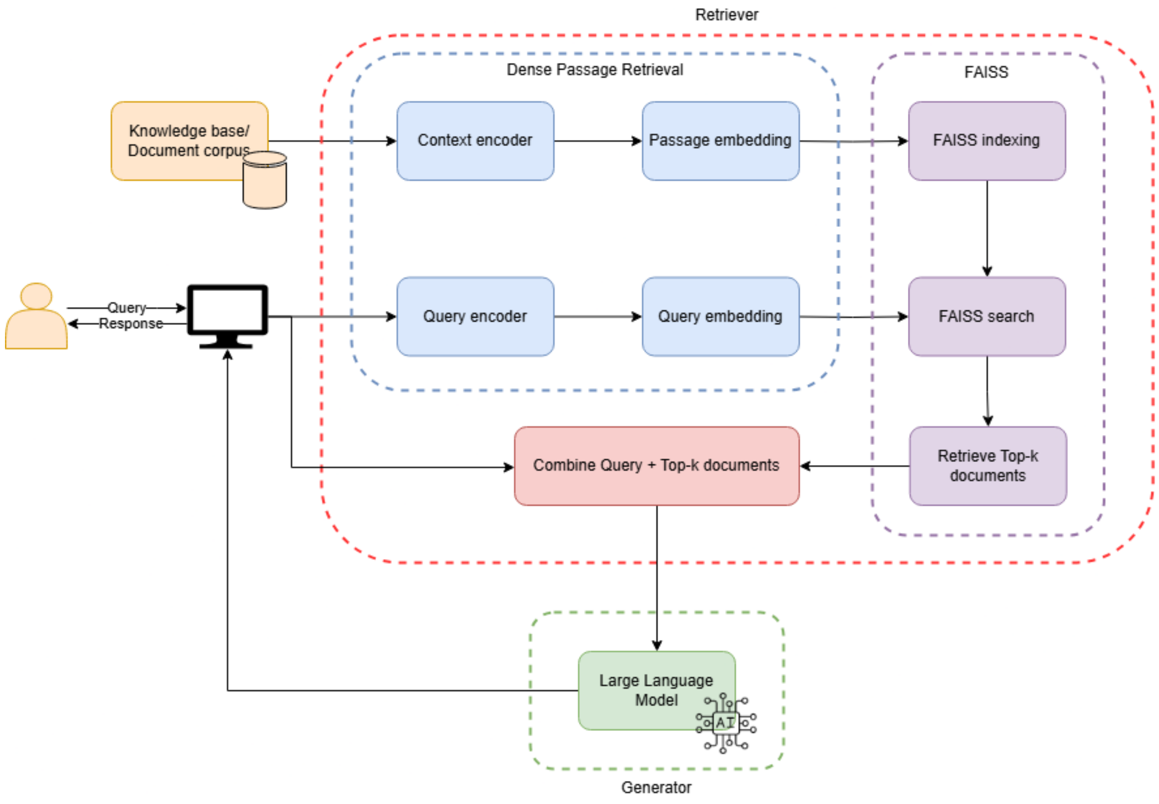

Fig. 3.6: RAG retriever architecture using Dense Passage Retrieval (DPR) and FAISS (adapted from [11, 30])

Figure 3.6 shows how DPR and FAISS work together as the retriever component in a RAG pipeline. The user query is first encoded into a dense vector using the query encoder. The document corpus has already been embedded using a separate context encoder. FAISS performs fast similarity search between the query and the indexed document vectors. The top-k results are returned and combined with the query. This combined input is sent to the language model, which generates the final output based on both the question and the retrieved external knowledge.

3.5.4 Benefits and Limitations

RAG systems significantly improve factual grounding by integrating external knowledge. Mansurova et al. [27] demonstrate that RAG increases robustness in question answering by reducing hallucinations and detecting knowledge gaps. However, Liu et al. [28] show that while RAG helps simplify reasoning tasks by retrieving intermediate evidence, it does not fully replace deep reasoning capabilities of LLMs. Moreover, retrieval quality is critical: noisy or irrelevant top-k results can mislead the generator, which underlines the importance of high-precision retrievers such as DPR combined with FAISS indexing.

3.6 Evaluation Metrics for Retrieval

3.6.1 Precision@k

Precision@k measures the fraction of retrieved documents that are relevant within the top-k results. It is defined as

$$\text{Precision@k} = \frac{\text{Number of relevant documents in top-}k}{k}.$$

This metric gives a straightforward indication of how many of the highest-ranked items are useful to the user. A known limitation is that it treats all relevant documents equally, regardless of their position. For example, a relevant document retrieved at rank 1 is valued the same as one retrieved at rank k, even though in practice higher-ranked results are usually more important to the user.

3.6.2 Normalized Discounted Cumulative Gain (nDCG)

Normalized Discounted Cumulative Gain (nDCG) evaluates both the relevance and the position of retrieved documents. The underlying idea is that highly relevant documents should appear early in the ranking. It is defined as [31]:

$$DCG@k = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)}, \quad nDCG@k = \frac{DCG@k}{IDCG@k}.$$

Here, rel_i denotes the graded relevance of the document at position i, while $IDCG@k$ is the maximum possible DCG if all relevant documents were perfectly ordered. The normalization ensures that nDCG scores lie between 0 and 1, where 1 corresponds to an ideal ranking.

3.6.3 BERTScore

While Precision@k and nDCG focus on retrieval relevance, BERTScore evaluates the semantic similarity of generated text against reference outputs. **BERTScore**, introduced by Zhang et al. [32], instead compares generated and reference texts in a semantic embedding space. It uses contextualized embeddings from a pre-trained BERT model to evaluate similarity between tokens.

Formally, let x be the reference sentence and y the generated sentence. Each token is mapped into an embedding $E(x_j)$ or $E(y_i)$. Precision is computed by checking, for every token in the generated text y, how well it matches the most similar token in the reference:

$$\text{Precision} = \frac{1}{|y|} \sum_{y_i \in y} \max_{x_j \in x} \cos(E(y_i), E(x_j)).$$

Recall is computed in the opposite direction, from reference tokens to generated tokens:

$$\text{Recall} = \frac{1}{|x|} \sum_{x_j \in x} \max_{y_i \in y} \cos(E(x_j), E(y_i)).$$

Finally, the F1-score balances both aspects:

$$F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

In essence, BERTScore aligns each token with its most semantically similar counterpart in the other sentence using cosine similarity. This allows it to capture meaning rather than exact wording. Because of this, it correlates strongly with human judgment and has become widely used in evaluating text generation tasks, including translation, summarization, and RAG-based generation.

3.7 Threat Intelligence Sources

3.7.1 Common Vulnerabilities and Exposures (CVE)

3.7.2 OpenASC Threat Catalog

3.8 Ansys Medini Analyze

3.9 Ollama

4 Methodology

This chapter outlines the methodological foundation of the project, including the development strategy, dataset composition, technology selection, structured attack tree design, and the system's architectural planning and evaluation metrics.

4.1 Development and Evaluation Strategy

- Iterative prototyping approach with continuous improvements
- Quantitative evaluation of system outputs

4.2 Dataset Preparation

- Dummy vehicle architecture modeled in Medini Analyze (ISO 21434-based)
- Manually created attack trees based on realistic automotive threats
- Integration of OpenASC JSON threat catalog as retrieval source

4.3 Evaluation of Retrieval Methods

- Comparison of retrieval strategies: TF-IDF, BM25, ColBERT, DPR + FAISS
- Final selection: Sentence-BERT (MiniLM) + FAISS for semantic threat matching

4.4 Evaluation of Generation Models

- Model comparison: GPT-4, Bart, DeepSeek, LLaMA
- Testing: LLaMA 3.1 8B (local) → Deployment: LLaMA 3.3 70B (quantized) on AWS via Ollama

4.5 Design of JSON-based Attack Tree Format

- Structure inspired by Medini Analyze (threat_scenario, attack_goal, attack_step)
- Rating dimensions and feasibility scoring rules
- Structural constraints to enforce validity and realism

4.6 System Design Overview

- Modular pipeline: Retriever, PromptManager, Generator, TreeBuilder, Validator
- Flow of data and model interaction across pipeline components

4.7 Evaluation Criteria

- Retrieval metrics: Precision@k, nDCG
- Generation metrics: BERTScore and rule-based tree validation

5 Implementation

5.1 Environment Setup

5.1.1 Local Development Environment

5.1.2 Virtual Environment and Dependencies

5.1.3 Ollama Setup and Local Testing

5.1.4 AWS Deployment of LLaMA 3.3 70B (g6e.4xlarge, L40S GPU)

5.1.5 Project Structure and Directory Layout

5.1.6 Launch Script and Execution Workflow

5.2 Data Handling and Preparation

5.2.1 Medini-Based Dummy Dataset

5.2.2 Custom Handcrafted Attack Trees

5.2.3 OpenASC JSON Catalog Integration

5.2.4 DataLoader Class and Preprocessing

5.3 Retrieval Pipeline Implementation

5.3.1 Embedding Model (MiniLM) Setup

5.3.2 FAISS Index Construction

6 Evaluation and Results

6.1 Example Outputs

6.1.1 Generated Attack Trees vs. Reference Trees

6.1.2 Observations on Semantic and Structural Alignment

6.2 Model Output Comparison (8B vs. 70B)

6.2.1 Consistency and JSON Validity

6.2.2 Depth and Richness of Attack Trees

6.2.3 Observed “Risk-taking” in Generated Paths

6.3 Retrieval Performance

6.3.1 Precision@k Results

6.3.2 nDCG Results

6.4 Generation Quality

6.4.1 BERTScore Results

6.4.2 Structural Validation Findings

6.5 Challenges and Limitations

6.5.1 Invalid JSON and Hallucinations

7 Conclusion and Future Work

7.1 Summary of Contributions

7.2 Critical Reflection

7.3 Outlook and Future Work

Bibliography

[1] Anay Prasanth Nair et al. "The Need for Cybersecurity in Automotive Industry". In: *Journal of Student Research* 13.2 (2024). DOI: 10.47611/jsrhs.v13i2.6789. URL: https://www.jsr.org/hs/index.php/path/article/view/6789.

[2] Sebastian Chlup et al. "THREATGET: Towards Automated Attack Tree Analysis for Automotive Cybersecurity". In: *Information* 14.1 (2023). ISSN: 2078-2489. DOI: 10.3390/info14010014. URL: https://www.mdpi.com/2078-2489/14/1/14.

[3] *ISO/SAE 21434:2021 — Road vehicles — Cybersecurity engineering*. International Standard. First edition, August 2021. Geneva, Switzerland and Warrendale, PA, USA, 2021. URL: https://www.iso.org/standard/70918.html.

[4] Marco Simoni et al. "MoRSE: Bridging the gap in cybersecurity expertise with retrieval augmented generation". In: *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*. 2025, pp. 1213–1222.

[5] Alan Birchler De Allende, Bastien Sultan, and Ludovic Apvrille. "Automated Attack Tree Generation Using Artificial Intelligence and Natural Language Processing". In: *International Conference on Risks and Security of Internet and Systems*. Springer. 2024, pp. 141–156.

[6] *Company History / Milestones*. 2024 (accessed July 2024). URL: https://www.continental.com/en/company/history/milestones/.

[7] Rodrigo Araújo Castro. "A semi-automated approach to UDS implementation testing". MA thesis. University of Porto, 2023 (accessed July 2024). URL: https://repositorio-aberto.up.pt/bitstream/10216/152123/2/636952.pdf.

[8] Joshua Dwight. "Building Cyber Attack Trees with the Help of My LLM? A Mixed Method Study". In: *Proceedings of the 2024 12th International Conference on Computer and Communications Management*. 2024, pp. 132–138.

[9] Tanmay Khule. "STAF: Leveraging LLMs for Automated Attack Tree-Based Security Test Generation". MA thesis. The University of Western Ontario (Canada), 2024.

[10] Mike Tanner. *Securing Vehicle Data: Implementing Standardization for Safe Access and Mitigating Cyber Threats*. Accessed: 2025-09-05. Auto Care Association. Aug. 4, 2023. URL: https://www.autocare.org/detail-pages/blog/et/2023/08/04/cars-are-the-next-attack-surface-frontier.

[11] Linta Joseph. "Automation of Threat Analysis and Risk Assessment (TARA) using Artificial Intelligence (AI)". MA thesis. Technische Hochschule Mittelhessen, 2025.

[12] Manfred Vielberth et al. "Elevating TARA: A Maturity Model for Automotive Threat Analysis and Risk Assessment". In: *Proceedings of the 19th International Conference on Availability, Reliability and Security*. 2024, pp. 1–9.

[13] Bruce Schneier. "Attack trees". In: *Dr. Dobb's journal* 24.12 (1999), pp. 21–29.

[14] Alyzia-Maria Konsta et al. "Survey: automatic generation of attack trees and attack graphs". In: *Computers & Security* 137 (2024), p. 103602.

[15] Sjouke Mauw and Martijn Oostdijk. "Foundations of attack trees". In: *International Conference on Information Security and Cryptology*. Springer. 2005, pp. 186–198.

[16] Yunpeng Wang et al. "A Systematic Risk Assessment Framework of Automotive Cybersecurity". In: *Automotive Innovation* 4 (Mar. 2021). DOI: 10.1007/s42154-021-00140-6.

[17] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, with Language Models*. 3rd. Online manuscript released August 24, 2025. 2025. URL: https://web.stanford.edu/~jurafsky/slp3/.

[18] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[19] N. Adaloglou. *Transformer Architecture Illustration*. https://theaisummer.com/static/6122618d7e1466853e88473ba375cdc7/40ffe/transformer.png. (Accessed on 04.09.2025). 2021.

[20] Kai Han et al. "Transformer in transformer". In: *Advances in neural information processing systems* 34 (2021), pp. 15908–15919.

[21] Markus Nagel et al. "A white paper on neural network quantization". In: *arXiv preprint arXiv:2106.08295* (2021).

[22] Dhairya Dalal et al. "Inference to the best explanation in large language models". In: *arXiv preprint arXiv:2402.10767* (2024).

[23] Ernesto Quevedo et al. “Detecting hallucinations in large language model generation: A token probability approach”. In: *World Congress in Computer Science, Computer Engineering & Applied Computing*. Springer. 2024, pp. 154–173.

[24] Ziwei Xu, Sanjay Jain, and Mohan Kankanhalli. “Hallucination is inevitable: An innate limitation of large language models”. In: *arXiv preprint arXiv:2401.11817* (2024).

[25] Ggaliwango Marvin et al. “Prompt engineering in large language models”. In: *International conference on data intelligence and cognitive informatics*. Springer. 2023, pp. 387–402.

[26] Ammar Mohammed and Rania Kora. “A Comprehensive Overview and Analysis of Large Language Models: Trends and Challenges”. In: *IEEE Access* 13 (2025), pp. 95851–95875. DOI: 10.1109/ACCESS.2025.3573955.

[27] Aigerim Mansurova, Aiganym Mansurova, and Aliya Nugumanova. “QA-RAG: Exploring LLM Reliance on External Knowledge”. In: *Big Data and Cognitive Computing* 8.9 (2024). ISSN: 2504-2289. DOI: 10.3390/bdcc8090115. URL: https://www.mdpi.com/2504-2289/8/9/115.

[28] Jingyu Liu, Jiaen Lin, and Yong Liu. “How much can rag help the reasoning of llm?” In: *arXiv preprint arXiv:2410.02338* (2024).

[29] ADAPALA CHANDRIKA PRIYANKA and DASARI RAMA KRISHNA. “Dual-Encoder Dense Retrieval Framework for Efficient Question Answering”. In: *INTERNATIONAL JOURNAL* 9.3 (2025).

[30] Hari Krishan Bekkam. *RAG Architecture: The Brains Behind Retrieval-Augmented Generation*. (Accessed on 06.09.2025). 2025. URL: https://miro.medium.com/v2/resize:fit:828/format:webp/1*muGP9N4dfrPainHq6i4Hdg.png.

[31] Kalervo Järvelin and Jaana Kekäläinen. “Cumulated gain-based evaluation of IR techniques”. In: *ACM Transactions on Information Systems (TOIS)* 20.4 (2002), pp. 422–446.

[32] Tianyi Zhang et al. “Bertscore: Evaluating text generation with bert”. In: *arXiv preprint arXiv:1904.09675* (2019).

Appendix

Tool	**Use Case**
Perplexity	- Literature review and summarization of scientific publications
Phind	- Quick research for technical questions - Support with programming tasks and problem-solving
ChatGPT	- Quick access to information across various topics - Assistance with programming tasks and code explanations
DeepL	- Translation of text passages between different languages (overall) - Improvement and correction of written texts for grammar and style (overall)

Table A1: Overview of the Use of AI-based Tools