



**Technical University of Munich**

School of Computation, Information and Technology

-- Informatics --

Bachelor's Thesis in Computer Science

**Automatic Term Extraction based on  
GenAI**

Jonas Ludwig Gerg



# **Technical University of Munich**

School of Computation, Information and Technology  
-- Informatics --

Bachelor's Thesis in Computer Science

## **Automatic Term Extraction based on GenAI**

Automatische Begriffsextraktion basierend auf  
GenAI

<b>Author:</b>	Jonas Ludwig Gerg
<b>Supervisor:</b>	Prof. Dr. Pramod Bhatotia
<b>Advisors:</b>	Prof. Dr. Bernd Brügge
<b>Start Date:</b>	24.03.2025
<b>Submission Date:</b>	24.07.2025

I confirm that this Bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 24.07.2025

Jonas Ludwig Gerg

## **Transparency in the use of AI tools**

I used DeepL to enhance language quality and translate parts of the thesis. I used ChatGPT to improve wording and sentence structure in order to optimize clarity. I have carefully checked all texts created with these tools to ensure that they are correct and make sense.

# Acknowledgements

I would like to express my sincere gratitude to Prof. Dr. Bernd Brügge for his invaluable guidance, support, and supervision throughout the course of this thesis.

My thanks also go to Prof. Dr.-Ing. Hans-Christoph Thiel for providing exemplary data on terminology in railway operations during the design and preparation phase, which significantly contributed to the quality and direction of this work.

I am also grateful to Deutsche Bahn for sharing their expertise and offering valuable insights into the topic, which helped shape the practical relevance of this thesis.

## **Abstract**

Deutsche Bahn is facing an increasing demand for a qualified workforce in various areas. Special terminology in railway operations makes it difficult for new and foreign workers to join the company. The emergence of new algorithms, models, and data sources necessitates a system capable of adapting to these innovations.

This thesis describes TAS (Terminology Agent System), a system based on the architectural blackboard pattern and Generative AI (GenAI), integrating Large Language Models (LLMs). TAS uses an event based system to coordinate various Knowledge Sources (KS), which are responsible for text extraction, text normalization, and definition generation. Its modular design enables the extension for new algorithms and external data sources.

A requirement analysis, grounded in use cases from Deutsche Bahn, was conducted using the OOSE methodology. The development process was based on Continuous Integration and Delivery. TAS was deployed to the Deutsche Bahn. A product test was carried out. The system achieved a recall rate of 91-98%, a precision rate of 78%, and generated usable definitions in 83% of the test cases.

Two unit tests and one integration test were conducted to evaluate TAS. The thesis discusses challenges arising from the non-deterministic nature of current LLMs. Threats to validity—such as varying testing conditions, choice of language, and the specific model used—are also examined.

## **Zusammenfassung**

Die Deutsche Bahn sieht sich mit einer steigenden Nachfrage nach qualifizierten Arbeitskräften in verschiedenen Bereichen konfrontiert. Die spezielle Terminologie im Bahnbetrieb erschwert neuen und ausländischen Mitarbeitern den Einstieg in das Unternehmen. Das Aufkommen neuer Algorithmen, Modelle und Datenquellen erfordert ein System, das sich an diese Innovationen anpassen kann.

Diese Arbeit beschreibt TAS (Terminology Agent System), ein System, das auf dem architekturellen Blackboard Pattern und Generative AI (GenAI) basiert und Large Language Models (LLMs) integriert. TAS verwendet ein ereignisbasiertes System zur Koordination verschiedener Wissensquellen (Knowledge Sources), die für die Textextraktion, Textnormalisierung und Definitionsgenerierung zuständig sind. Der modularer Aufbau ermöglicht die Erweiterung um neue Algorithmen und externe Datenquellen.

Eine Anforderungsanalyse, basierend auf Anwendungsfällen der Deutschen Bahn, wurde unter Verwendung der OOSE-Methodik durchgeführt. Der Entwicklungsprozess basierte auf Continuous Integration und Delivery. TAS wurde bei der Deutschen Bahn eingesetzt. Es wurde ein Produkttest durchgeführt, in dem das System eine Recall-Rate von 91–98% und eine Präzisionsrate von 78% erreichte und generierte in 83% der Testfälle brauchbare Definitionen.

Zur Validierung von TAS wurden zwei Unit-Tests und ein Integrationstest durchgeführt. Die Arbeit diskutiert Herausforderungen, die sich aus der Nicht-Deterministik aktueller LLMs ergeben. Auch Bedrohungen für die Validität – wie unterschiedliche Testbedingungen, die Wahl der Sprache und das verwendete spezifische Modell – werden untersucht.

# Contents

<b>1 Introduction</b>	<b>2</b>
1.1 Motivation	2
1.2 Objectives of this Thesis	3
<b>2 Background</b>	<b>3</b>
2.1 Automatic Term Extraction	3
2.2 Challenges with ATE	5
<b>3 Requirements Analysis</b>	<b>6</b>
3.1 Overview	6
3.2 Scenarios	7
3.3 Requirements	9
3.4 System Models	11
<b>4 System Design</b>	<b>19</b>
4.1 Overview	20
4.2 Design Goals	20
4.3 Subsystem Decomposition	22
4.4 Hardware / Software Mapping	25
4.5 Continuous Integration and Delivery	26
4.6 Persistent Data Management	27
4.7 Access Control	27
4.8 Global Software Control	27
<b>5 Object Design</b>	<b>29</b>
5.1 Annotated Scenarios	29
5.2 Prompt Engineering and Application of LLMs	31
5.3 Knowledge Sources	32
<b>6 Extensibility of TAS</b>	<b>37</b>
6.1 AI models - BahnGPT	38



6.2 Implementation of new algorithm .....	39
6.3 Implementation of new data source .....	40
<b>7 Testing .....</b>	<b>42</b>
7.1 Unit Test .....	42
7.2 Integration Test .....	44
7.3 System Test .....	48
7.4 Difficulties in testing applications using LLMs .....	50
7.5 Threats to validity .....	51
<b>8 Conclusion and Future Work .....</b>	<b>52</b>
8.1 Conclusion .....	52
8.2 Future Work .....	53
<b>List of Figures .....</b>	<b>56</b>
<b>Appendix A: Supplementary Material .....</b>	<b>59</b>
<b>Appendix B: Test Results .....</b>	<b>60</b>
<b>Appendix C: Source Code .....</b>	<b>60</b>
<b>Bibliography .....</b>	<b>61</b>

## Typographical Conventions

Throughout this thesis the following conventions are used:

- Citations are given in a comprehensive form (e.g. [ABC08]), indicating the first three authors (e.g. Alpha, Beta, Caesar) of an article by capital letters followed by the year of publication (2008). If a “ + ” appears in the citation, then more than three authors have contributed and only the first author will be mentioned by their last name: the first letter in capitals followed by two further lower case letters (e.g. [Del+24]). In case of a single author, the first letter of his last name is written in capitals, followed by two further lower case letters (e.g. [Cal03]). If multiple sources exist with the same authors and year of publication, additional letters are added to the citation (e.g. [DB25b]) to make the reference unique.
- Related work is given in form of *inline citations* wherever needed, instead of reporting existing research in a monolithic block at the beginning or end of a chapter.
- The *Unified Modeling Language* (UML) is used for the illustration of abstractions and for the modeling of software components.
- Abstractions like classes and use case names are highlighted in **bold font**.
- Upper capitals are used for product names, such as DOCKER or SPACY.
- Implementation and code snippets are rendered in `monospace` font.
- Codeblocks are rendered using the Typst CODLY package is used for this.
- Technical terms are written in *italics* when they appear the first time.
- Aliases in models are marked in red

# 1 Introduction

## 1.1 Motivation

Deutsche Bahn is facing a strong demand for qualified specialists in various areas: as of begin of March 2025, around 3000 open fulltime positions are listed in the job portal of DB, ranging from technicians and construction plant operators to skilled drivers [Deu25a]. Around 400 bus drivers, some of whom were recruited from various European countries such as Poland, Spain and Croatia, were deployed for the rail replacement service alone during the renovation work on the Riedbahn line [Deu24a]. The Group is increasingly focussing on recruitment from abroad [dpa24, Til22]. Employees are required to obtain at least level B1 of the languages of the respective European countries they are deployed in, for apprenticeship and study programmes often a higher level is needed [Deu25b]. Moreover, cross-border train service requires train drivers to possess skills in the respective foreign language [Eur19], which limits the flexible use of workers for different routes as Germany is adjacent to six countries with languages other than German. For subsidiaries at Deutsche Bahn, providing language programs is expensive as the course themselves are costly and cause absence of staff. For example, a language course for German A1 at Goethe Institut in Berlin costs roughly 800€ per participant and requires 50 to 70 teaching units [Goe25]. Since generally B1 is required for transborder traffic, completing further education programs is even more time consuming.

To reduce potential language barriers, the “DB Translator KITT” [DB 25a] was developed — a translation tool specifically trained on railway-related language. This application is intended to be extended by a learning module that displays, translates, and defines the terms and phrases used during communication. The core challenge lies in how to automatically and accurately extract these terms and phrases from a given text. The system should also be able to adapt to different subdomains in order to generate fitted results for different contractors. For example, the subsidiaries of Deutsche Bahn DB InfraGO might have different terminology set, as well as different documents available to them compared to others, for example instruction manuals.

## 1.2 Objectives of this Thesis

This thesis addresses the problem of Automatic Term Extraction (ATE) and explores how this and related text analysis tasks can be approached using Generative AI (GenAI), especially using Large Language Models (LLMs). The focus of this thesis is on the proposed system, the Terminology Agent System (TAS), and its extensibility and flexibility.

Chapter 2 provides an overview of Automatic Term Extraction (ATE) methods and addresses its challenges. In Chapter 3, requirements for a flexible system supporting ATE and text analysis are derived based on real-world scenarios at Deutsche Bahn. Furthermore, corresponding system models are developed. Chapter 4 defines the design goals and outlines the overall architecture of the TAS, including its interaction with external components. Chapter 5 focuses on the implementation of the Knowledge Sources (KS) and the application of prompt engineering strategies. Chapter 6 demonstrates the extensibility and flexibility of the system. Chapter 7 discusses testing approaches for applications involving large language models (LLMs), including challenges that arise in this context. Chapter 8 concludes the thesis by summarizing the system, evaluating achieved and unmet goals, and providing an outlook on future work.

## 2 Background

This chapter provides an overview of Automatic Term Extraction (ATE), a process in natural language processing that identifies and extracts domain-specific terms from text. Section 2.1 explains the process of ATE and explores various approaches, including traditional and recent methods. Section 2.2 discusses challenges of Automatic Term Extraction and implications for the choice of ATE method.

### 2.1 Automatic Term Extraction

Automatic Term Extraction (ATE) involves the identification and extraction of terminology that is specific to a particular domain within a document using

an automated process [Tra+23]. ATE is used in traditional linguistic research and terminology science, focusing on determining domain specific terminology, as well as the computer linguistic task of information retrieval, mostly concerning the indexability of documents based on its semantic meaning [DMS23]. Various approaches exist to address this challenge [Mit22]:

**Linguistic:** Linguistic approaches analyze properties of the sentence structure like Part-of-Speech (POS), syntax and grammatical rules. Terms are subsequently identified based on predefined patterns. However, this approach is highly dependent on the specific language being analyzed, which can limit its applicability across different languages. For example, the LEXTER [Bou92] first runs a POS tagger on a given text and then extracts phrases based on predefined patterns.

**Dictionary-based:** Dictionary-based approaches utilize ontologies and glossaries to match known terms within texts. These methods require a manually curated set of terminology, which is expensive to develop and maintain. A simple implementation can be achieved using Regular Expressions by identifying matching terms within a text.

**Statistical:** Statistical methods rely on frequency analysis of tokens in documents and domains. For example, a set of tokens is considered a term of a domain if it occurs frequently within a document or a domain compared to others. One example of a statistical method is the C-Value algorithm, which favors term candidates that are nested rarely in other term candidates [FAM00].

**Large Language Models:** Large Language Models have a deep understanding of language and are capable of solving a range of NLP tasks [Qin+24], which makes them well-suited for term extraction. These models can facilitate direct extraction: for instance, models like BERT can be fine-tuned for term extraction, or prompt-based models like GPT-4 can be instructed to extract terminology directly.

NLP research is increasingly focusing on the use of large language and transformer based models like GPT and BERT [BCM24], as well as prompt based approaches [Gig23], as they show promising improvements in cross-language task

solving and overall effectiveness in term extraction when compared to traditional methods.

[Tra+22] conducted experiments with pretrained transformer models on English, French, and Dutch corpora containing domain-specific terminology, reporting recall scores ranging from 50% to 75%. [Haz+22] employed fine-tuned BERT models and hybrid approaches combining linguistic and statistical filters for automatic term extraction across various languages and domains, achieving recall rates of up to 82%. In a prompt-based approach, [Gig23] evaluated the performance of GPT-4 using a short prompt, comparing it to a statistical phrase-based term extraction method. While the statistical approach yielded higher recall rates, it produced a disproportionately high number of irrelevant or incorrect terms. In contrast, the LLM-based method demonstrated lower recall but substantially higher precision.

Although ATE tools aim to produce high-quality results, manual verification is still necessary, as current methods do not yet offer fully reliable performance [Haz+22, DMS23]. Even state-of-the-art transformer-based models often report recall scores below 80% and precision rates under 60% [Haz+22].

## 2.2 Challenges with ATE

According to [Mit22], there are four major challenges with automatically extracting terminology from texts.

**Domain dependency:** ATE systems often require customization for different domains. Especially dictionary-based approaches perform worse when solutions for one domain are used in others. Statistical approaches tend to adapt better to different domains but benefit from custom linguistic filters.

**Variability of Terms and Polysemy:** The same meaning or concept can be represented by different terms, leading to variability in terminology. Furthermore, a term may be polysemic, which refers to its ability to represent multiple concepts: for example, the German word “Bank” can denote both a seat and a financial institution. Additionally, abbreviations and orthographic differences (e.g. “e-mail” instead of “email” or typos) constitute another form of term variation. Dictionary-

based approaches suffer from its lexical limitations. Syntactic variations and terms that are spread over a sentence are especially difficult for pattern based methods. For example, the term “auf Sicht fahren” can be used in different variations and positions within a sentence: “Ich muss hier *auf Sicht fahren*”, “Hier gilt *Fahrt auf Sicht*”, “*Fahre* hier 10km *auf Sicht*”.

**Cross-Language Compatibility:** Dictionary-based approaches require the development of new dictionaries for each language, which can be resource-intensive. Additionally, linguistic filters and syntactic rules must be tailored to accommodate the specific grammar of each language. In contrast, statistical and hybrid methods tend to be more language-independent, allowing for broader applicability across different languages.

**Scalability:** Dictionary-based methods face challenges with large dictionaries, particularly concerning lookup times, which can impede efficiency. While LLM based approaches can combine multiple steps in hybrid approaches, cost and scalability can pose significant challenges.

### 3 Requirements Analysis

Based on the design process as described in [BD10], this chapter examines the requirements and scenarios as well as the resulting system models for the TAS. Section 3.1 provides a short overview about the purpose, scope and objectives of the system. Section 3.2 illustrates scenarios at Deutsche Bahn and their subsidiary in which the system may be used as well as its participating actors. Section 3.3 presents the functional and nonfunctional requirements for the system. Based on the previous insights, system models are derived in Section 3.4 and more details about the system are illustrated.

#### 3.1 Overview

The goal is to design and implement an extensible automatic terminology extraction system that incorporates domain specific data as context. It can be easily extended to also generate other linguistic features, like definitions or term

relations. New algorithms or data sources should be easy to integrate and the system should be reusable for other use cases. The final goal is to have a system that extracts important linguistic features and definitions for workers to improve their language skills and help understand instructions.

## 3.2 Scenarios

The following sections illustrate scenarios in which the system may be used and its participating actors.

### 3.2.1 A conversation at the construction site

Construction worker **John** was hired from England and possesses B2 German language skills. He formerly worked in the construction industry and is not used to special terms used at DB InfraGO<sup>1</sup>. At the construction site, the colleague **Sabine** asks **John** in German to check the “Walzlänge” of the provided rails. As **John** does not fully understand the request, he takes the DB Translator KITT App, an internal translation service at DB, and lets his coworker repeat the sentence into the app. The input is translated into English. Worker **John** reads the translation, yet still has not heard of the term “mill length”. He presses on the text and the app shows all the terms that were recognized in the sentences in both languages German and English including their verified definitions. Worker **John** now finds the word “mill length” and its explanation and is now able to fully perform the coworkers request.

### 3.2.2 Understanding instructions

The Hungarian constructor **Gábor** of an ICE 3 train recognizes a defect on the on-board toilet. He looks up the respective instruction book on how to fix the clogged pipe. As he reads the instructions, some terms, for example “Fäkalienbehälter”, are unknown to him, so he uses the DB Translator KITT App to translate the sentence, in which the term occurs. The translation alone is not enough to fully comprehend the meaning of the sentence, since he doesn’t know the term

---

<sup>1</sup>As a subsidiary of Deutsche Bahn, DB InfraGO AG is responsible for building, operating, and maintaining Germany’s railway infrastructure [Deu25c].



“székletgyűjtő tartály”<sup>2</sup>. He taps on the translation to get an overview of contained terms and finds the unknown term. He reads the definition of the term and is now able to fully understand the instructions.

### 3.2.3 Learning Richtlinien<sup>3</sup>

Train driver **Louise** is to be deployed on the route between France and Germany for the next timetable change. She already has the required language qualifications, but needs to revise some of the rules for rail services in Germany. Therefore, she gets the official rulebook Ril 302.6000 [DB 24a] from Deutsche Bahn. She reads the following sentence: “Streckenfernsprechleitung (Zugmeldeleitung) zwischen Fdl Bantzenheim und Fdl Neuenburg. Der Fdl Bantzenheim kann den Fdl Neuenburg über die Taste „Cantonnement Neuenburg“ erreichen.”. However, she does not fully understand all the words and their definitions.

She opens the DB Translator KITT App on her phone, enters the sentence and taps the “Translate” button. After the translation is shown on the screen, she taps on it and sees the term “Streckenfernsprechleitung” and “Zugmeldeleitung” and the French translation, which is “circuit téléphonique omnibus”. Underneath the term “Streckenfernsprechleitung”, she reads the official definition and continues her work.

### 3.2.4 Gaining deeper knowledge

This scenario continues on Section 3.2.2 “Understanding instructions”. Later **Gábor** finds a new problem on the train. There is a blinking light on a control board and he consults the respective instruction book and the respective page. However, he is unable to fully understand the German instructions provided in the handbook. He enters the instructions into the translation app and receives a German translation. He wants to know more about the problem, that is described, so he asks where else the instruction is found. A list of references to other instruction books is displayed to the user. He saves the references for later by pressing on a bookmark icon.

---

<sup>2</sup>Hungarian translation for /"Fäkalienbehälter".

<sup>3</sup>“Richtlinien” are the official guidelines and rulebooks at Deutsche Bahn.

### 3.2.5 Find new vocabulary used with coworkers

The linguist and educator **D** is focusing on how terminology evolves in his sector in rail construction. He is given a set of conversations from other employees that agreed to share their utterances with him. He uploads the document, along with other instruction books and regulation PDFs to a website. After a while, the website shows him a list of terms that were found in the input and were assumed to be especially relevant for the railway domain. For some terms, official definitions from Deutsche Bahn are shown, for others, there is no verified definition, but an automatically generated definition based on the context of the conversation. He finds the word “Betra” with a presumed definition of it being used in context of building and regulations. The website suggests, that the actual term is “Betriebs- und Bauanweisung”. After further investigation, he confirms the guess and inserts the new term into the companies glossary of verified terms.

## 3.3 Requirements

In the following section, the functional and nonfunctional requirements are described based on the FURPS+ model<sup>4</sup>. These also include requirements and constraints from the contractee Deutsche Bahn.

### 3.3.1 Functional Requirements

- FR 1    Extract domain terminology in a sentence:** It ignores terminology that is not relevant to the respective domain.
- FR 2    Recognize multi word terminology:** Terms that are spread over a sentence are properly recognized.
- FR 3    Lemmatize terminology:** Terms should be lemmatized to its base form.
- FR 4    Retrieve predefined definition:** Human-verified definitions should be preferred if available.

---

<sup>4</sup>The FURPS model classifies software requirements into five categories: Functionality, Usability, Reliability, Performance, and Supportability [Gra92]. It is often extended as FURPS+ to include additional non-functional aspects such as design constraints and implementation requirements [BD10]

- FR 5     Generate definitions for terms with no verified definition:** If official definitions are not available, new definitions should be generated artificially based on the given context.

### 3.3.2 Nonfunctional Requirements

- NFR 1     Supportability: Extensibility of Knowledge Sources:** It should be easy to integrate new algorithms, AI models, data sources and Knowledge Sources at any time.
- NFR 2     Supportability: Reusability:** It should be easy to use the system for applications other than term extraction and text analysis, like ontology creation.
- NFR 3     Supportability: Multiple languages:** The system recognizes terminology not only in a single language and can be adapted to support languages other than German, without having to redesign the system.
- NFR 4     Adaptability: Cross-Domain compatibility:** The system should be adaptable to other domains without having to fine tune pretrained models for a specific use case.
- NFR 5     Performance: Reliability of term extraction:**  
The recall score for term extraction should be above 80%. The precision score of the term extraction should be above 60%<sup>5</sup>. In other words, the rate of terms that were expected and extracted by TAS, should be high and the rate of wrongly identified terms or irrelevant terms should be moderately low.
- NFR 6     Performance: Reliability of term definitions:** Artificially generated definitions must uphold factual accuracy and content that could lead to misunderstandings or actions that potentially harm users or others, should be absent in the generated text.

---

<sup>5</sup>These values were determined by the client, Deutsche Bahn.

- NFR 7      Performance: Term extraction:** When requesting term extraction and definition generation for a text, the result should take less than 10s [Jak25] to generate to retain the flow of work. For use at Deutsche Bahn, the request must be returned in under 3 seconds [DB 25b].
- NFR 8      Performance: Multiple users:** Multiple users should be able to request terminology in parallel.
- NFR 9      Usability: AI transparency:** AI generated definitions have to be visibly marked as such.
- NFR 10     Usability: Seamless interaction experience:**<sup>6</sup> Looking up terminology in a text should not affect work. The separate learning functionality of TAS should not impair the ease of use when translating conversation with other users.

### 3.4 System Models

This section describes the system models of TAS. Section 3.4.1 describes the functionality of the system with a use case model. Section 3.4.2 shows an analysis object model of TAS. Section 3.4.3 explains the dynamics of the system using an activity diagram. Section 3.4.4 shows the user interface that accesses the TAS.

#### 3.4.1 Use Case Model

Figure 1 shows the extracted usescases and actors from the scenarios in Section 3.2. The acting participants are

- a *Worker*, which in this case represents a employee at Deutsche Bahn (e.g. a construction worker)
- a *Linguist*, who works in language management at a company or a scientist analyzing domain terminology

---

<sup>6</sup>Empirical user studies are required to assess this requirement.

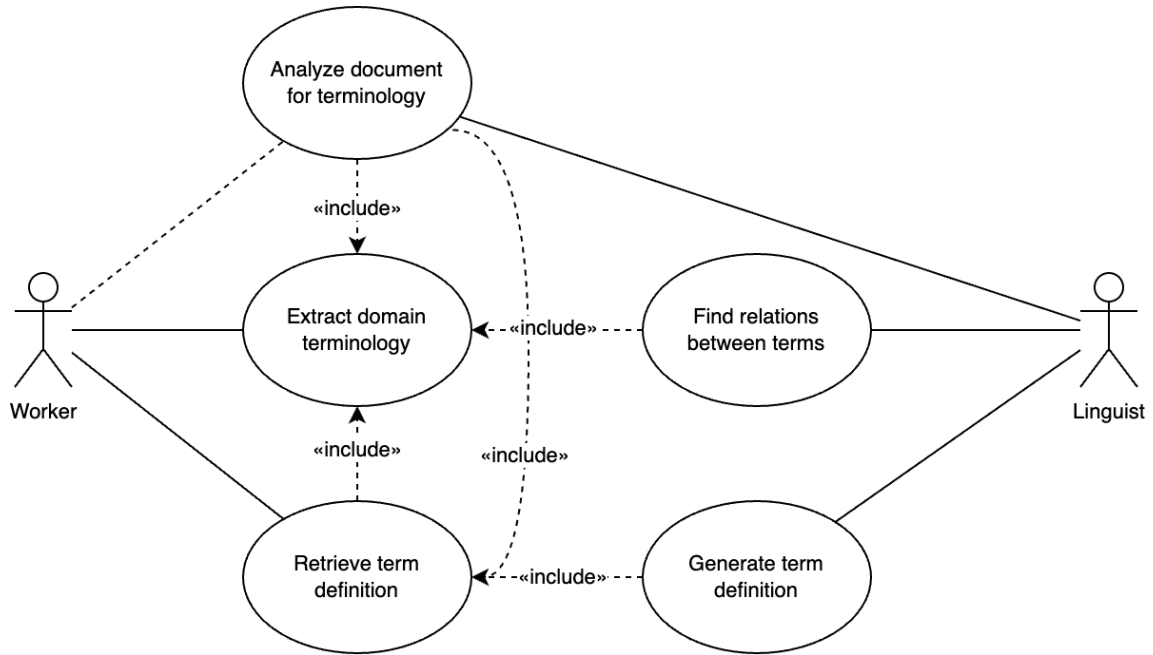


Figure 1: The functional model of TAS (UML Use case diagram)

---

In most scenarios, the `Worker` initializes the `Retrieve Term Definition` use case, when they require the definition of a term in a text. Therefore, the `Extract domain terminology` use case (see Figure 2) is also initialized. In special cases, for example if a whole PDF document is required to be analyzed, a `Worker` might initiate the `Analyze document for terminology` use case. A `Linguist` is interested in analyzing whole files or documents with multiple pages, which includes extraction and definition retrieval. Especially in case of a stakeholder in language quality assurance, the generation of definitions might be especially relevant, as they have the duty to validate definitions for terms used at their domain. Also, it might be interesting for the `Linguist` to `Find relations between terms`. This, again, requires terms to be extracted from a document or text, so that enough context is available to infer semantic relations between terms and their meanings.

<i>Use case name</i>	The Extract domain terminology use case.
<i>Participating actors</i>	Worker
<i>Entry condition</i>	Conversation has started, Utterances and translations have been made, document is available in text form
<i>Flow of events</i>	<ol style="list-style-type: none"> <li>1. The <b>Worker</b> presses the conversation text in the DB Translator KITT app.</li> <li>2. The event <b>TextExtracted</b> is published to the Blackboard. It includes the transcription of the respective conversation.</li> <li>3. The <b>TermExtractor</b> reacts to the event, activates and starts extracting terminology from the text.</li> <li>4. The <b>TermExtractor</b> finds a term and publishes the <b>TermExtracted</b> event.</li> <li>5. The <b>TermExtracted</b> events activates the <b>TermNormalizer</b>, which normalizes the detected term.</li> </ol>
<i>Exit condition</i>	All Knowledge Sources finishes execution

Figure 2: The Extract domain terminology use case. Template from [BD10]

### 3.4.2 Analysis Object Model

Figure 3 shows the object model of the terminology agent system.

The user interacts with the system through a **UserInterface**. In this thesis, it is realized as a module in an **App**, but it can also be a textual UI like a **CLI** or even a verbal interface like a **PhoneCall**. The **UserInterface** has a reference to a **Session**, that holds **Conversations** from previous interactions and the current one. It again consists of **Documents**, which can be multiple **Utterances** (for example see Figure 7) with different **Modality** (**Text**, **Video** or **Speech**), or other types of documents, like a PDF **File**. The user interacts with the Session through **Requests**, which can include translating an utterance, extracting terminology from a text, or asking a question about a document. **Documents** can also be stored in and retrieved from an arbitrary **DocumentManagement**.

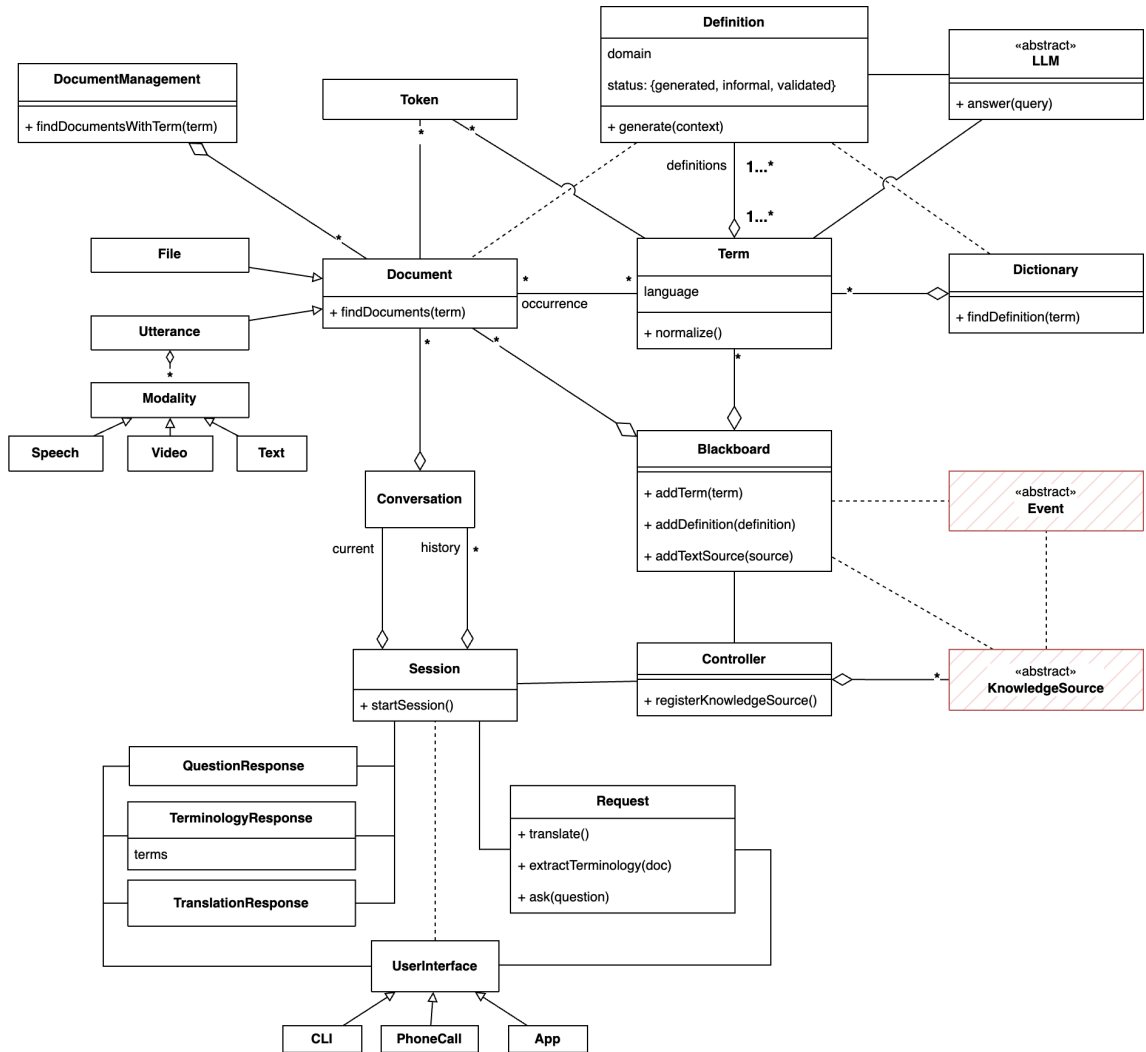


Figure 3: Analysis object model of TAS (UML class diagram)

Each **Document** has a plain text representation in the form of multiple **Tokens**<sup>7</sup>, which together can form a **Term**. Each term can consist of multiple tokens (e.g. “auf Sicht fahren”), belongs to a language and can be normalized, for example into a lemmatized<sup>8</sup> form. A term can have one or multiple **Definitions**, since it

<sup>7</sup>**token**: “Tokenization generally refers to the process of breaking up sequences of symbols into subsequences that can be represented as units, known as tokens.” [Gas+25] In this thesis, a **Token** refers to strings of characters separated by a whitespace.

<sup>8</sup>**lemma** “all the forms (base form and inflected forms collectively) of a word, usually cited as the base form and taken as being representative of all the various forms of a morphological paradigm” [Mit22]

might be polysemy<sup>9</sup>, or have different definitions based on the domain<sup>10</sup>. Definitions represent the senses of a term. They belong to a certain domain or topic and its status depends on whether the definition is artificially generated, marked as informal language or is verified and preferred by official stakeholders. Translations for a term can be retrieved indirectly by looking up terms that have the same definition instance. Definitions can also be generated based on additional context. They might use specialized topic modeling services to infer the topic of the context or use **LLMs** to generate definitions. If a definition is generated artificially, it might reference a **Document** that was used as context. A **Dictionary** stores those terms and definitions persistently and can retrieve them by term.

Analogous to the blackboard pattern [Lal97], the **Controller** contains a set of **KnowledgeSources** that are explained in detail in Figure 5. The controller activates one of the Knowledge Sources which then update the state of the **Blackboard**. The blackboard stores a list of **Terms**. The Knowledge Sources may lookup definitions from a **Dictionary**, generate them artificially or use the **DocumentManagement** to search for documents.

For later use, TAS should be able to maintain state throughout the current user **Session** and track previous user input as history. This can be used by other Knowledge Sources to gain more context over the current use of the system.

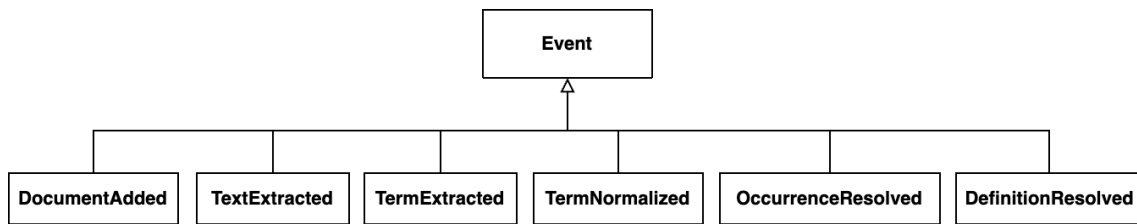


Figure 4: Event taxonomy of all events known by TAS (UML class diagram)

---

This system includes an event driven architecture, which follows the Observer pattern [BD10]. **KnowledgeSources** can publish **Events** during execution, which

---

<sup>9</sup>**polysemy**: A polysemic word can have multiple senses [Mit22], e.g. the German word “Bank” might mean a financial institution or a bench

<sup>10</sup>for example, the abbreviation “BR” can stand for “Baureihe für Triebfahrzeuge”, “Betriebsreinigung”, “British Railways”, “Betriebsrat” [Thi21].



are shown in detail in Figure 4. The **DocumentAdded** event contains the path to a pdf file. The **TextExtracted** event contains extracted text and its originating **Document** object. The **TermExtracted** event contains the extracted **Term** object as parameter. The **TermNormalized** is fired, when a **Term** object received a normalized version. The **OccurrenceResolved** event contains a **Term** and a **Document**, in which the term is found. The **DefinitionResolved** event contains the **Term** object and the generated **Definition**.

During execution, **KnowledgeSources** publish events, which other Knowledge Sources then listen to and therefore activate themselves.

Figure 5 shows the object model of the Knowledge Sources for the terminology blackboard. It consists of superclasses that denote the linguistic tasks performed on the blackboard, and their concrete implementations, which can be swapped easily.

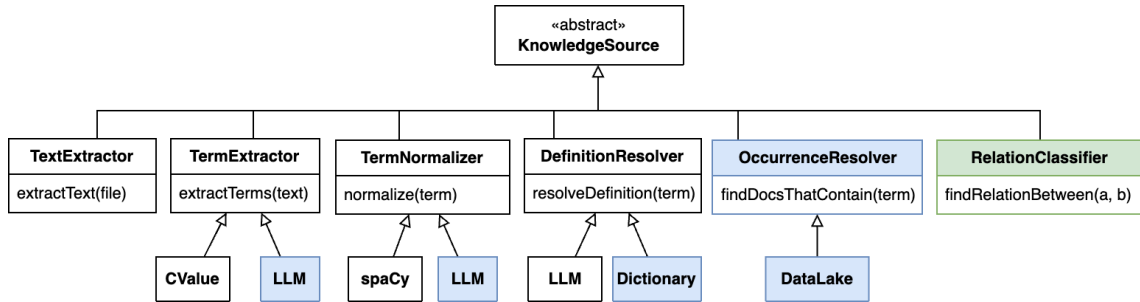


Figure 5: Taxonomy of all Knowledge Sources responsible for linguistic feature extraction (UML class diagram).

---

The **TextExtractors** task is to support multiple file types like PDFs or Word documents and extracts the plain text out of them. The **TermExtractor** is responsible for finding and filtering relevant term candidates in a document. The **TermNormalizer** tries to add variants of a term or find its base form (e.g. “Triebfahrzeugführerin” instead of “Triebfahrzeugführer”), which might be easier to process for other Knowledge Sources like the **DefinitionResolver**.

The **DefinitionResolver** annotates term candidates with definitions based on the context of the term. For this, LLMs will be used to generate text and a **Dictionary** is used to query existing, verified definitions.

The **OccurrenceResolver** is responsible for finding documents that match a given term or keyword, in order to gather more information for a unknown term. The **RelationClassifier** finds relations between recognized terms or even terms present in an existing ontology.

The abstract Knowledge Sources are designed to represent a single linguistic task and do not depend on a concrete implementation or algorithm. This enables easy extension of the system when new data sources for the domain become available or when new features like generating relations between terms are required. Multiple implementations of a Knowledge Source can also be deployed at once to improve the systems output quality.

### 3.4.3 Dynamic Model

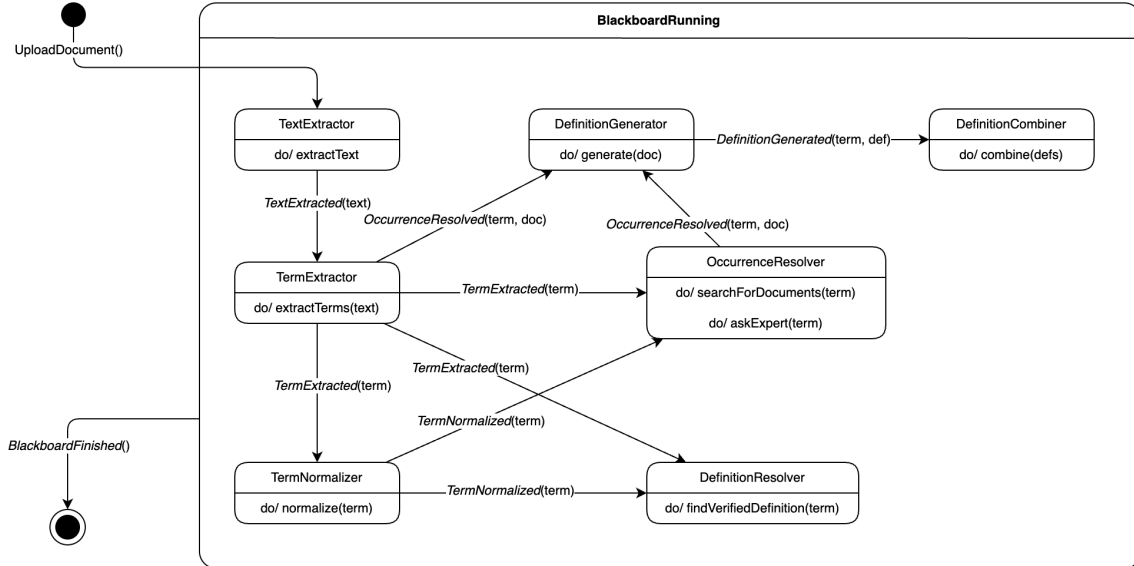


Figure 6: The flow of events of the blackboard system (UML activity diagram).

Figure 6 shows the dynamic model of the blackboard. The process starts when a document is uploaded, which publishes an initial event on the blackboard. In

general, the entry condition for a specific Knowledge Source is fulfilled, when an event the Knowledge Source listens to is dispatched on the blackboard. The controller then activates one or more Knowledge Sources that listen to the posted event. After a Knowledge Source completes, its results are posted on the blackboard and one or more events are published. The exact change on the blackboard made is passed in the event as arguments. Once there is no running Knowledge Source and no more events can be handled, the blackboard finishes.

This architecture combines the blackboard pattern and an event dispatcher following the observer pattern [BD10], which allows for multiple Knowledge Sources to be activated at once and still react to changes from other Knowledge Sources.

Each Knowledge Source publishes and reacts to one or more events. It is important to note that this blackboard system does not necessarily leave its current state, when a new event is published, and might be in multiple states at the same time. For example, the **TermExtractor** can fire the *TermExtracted* and *OccurrenceFound* event, once a single term is yielded. Both the **OccurrenceResolver** and the **DefinitionGenerator** can then start their process independently while the **TermExtractor** continues extracting more terms.

### 3.4.4 User Interface

Figure 7 shows the UI of the existing translation screen of DB Translator KITT [DB 25a] (left) and the new screen added showing the results of the TAS after requesting terminology information from a conversation (right). After the user of the app has started a conversation with another person and already made utterances, the app shows the respective transcriptions and translations on the screen. Newly introduced in this thesis, the option to long-press one of the text bubbles will be added, which loads and shows the terms and respective definitions contained in the selected transcription. The example on the right in Figure 7 shows the proposed UI for displaying the terminology. For example, the TAS recognized the term “Nachjustierung” in the sentence. Since there was no officially validated definition for it, an artificially generated definition is shown, which is annotated

with a ✨ (star). A message below the input text also informs the user that the definitions are generated using AI and may contain errors (NFR9) . If multiple definitions can be found for a term, they will be listed below. This can be the case if the TAS is not able to infer the topic of the conversation and therefore cannot select the correct definition.



Figure 7: Screenshots of the user interface for the existing DB Translator KITT (left) and the additional terminology information from TAS (right)

## 4 System Design

This chapter maps concepts of the application domain in Chapter 3 to the solution domain. Section 4.1 provides a brief overview of the software architecture and existing systems, as well as constraints impacting the design decisions. In Section 4.2, design goals are derived from nonfunctional requirements in Section 3.3.2 and prioritized. Section 4.3 describes the architecture of TAS by decomposing it into

subsystems. Section 4.4 shows the communication between the subsystems and their deployments and maps the subsystems to device nodes. Section 4.5 describes the process of continuous integration and delivery. Persistent data management and access control are briefly presented in Section 4.6 and Section 4.7. Finally, the global software control is described in Section 4.8 including an example event flow of TAS.

## **4.1 Overview**

TAS is primarily designed to be used with other systems at Deutsche Bahn and should be integrated into the existing translation app called “DB Translator KITT” (DBTK). The service consists of the DBTK ANDROID app and a PYTHON backend, that handles the translation process.

Also, the use of Generative AI is highly regulated at Deutsche Bahn and is subject to strict requirements, with only a small set of verified tools allowed for use in production. Therefore, the production version at Deutsche Bahn is constrained to only use an internal LLM service called BahnGPT [DB 25c], which will be introduced later.

## **4.2 Design Goals**

Derived from the nonfunctional requirements in Section 3.3.2 and from requirements of the Deutsche Bahn, a set of design goals for TAS is defined.

### **4.2.1 Performance**

While requirements for performance depend on the concrete use case TAS is used in (see Section 3.4.1), for this project a quick response time is required to maintain a good experience for the user while using the app (NFR5) .

### **4.2.2 Reliability and Safety**

Learning terminology correctly is the main focus of this thesis. Thus having reliable and consistent, artificially generated output is very important. Definitions have to be exact and not deviate from the actual ground truth meaning (NFR6) (low rate of false positives) so that the safety of workers it not jeopardised.

### 4.2.3 Extensibility

Since LLMs and other models improve very quickly [Ho+24], TAS should be designed to allow new Knowledge Sources to be integrated easily. New algorithms should be easy to implement and replace others (NFR1) .

Even though the main focus of this thesis lies on terminology extraction and definition generation, TAS should allow for new linguistic features and use cases, for example translation, without having to modify the underlying agent system (NFR2) .

### 4.2.4 Usability

For TAS to be successful, the app should not hinder users in production environments, but support them in understanding tasks and conversations (NFR10) . Therefore the user should receive as much information as possible without further interaction.

### 4.2.5 Trade-offs and prioritization

For the system to be used in working areas safely, **reliability and safety** are the most important design goals. These are measured manually and the goal is evaluated in Chapter 7. Especially the generation of definition has to be as precise and correct as possible. Following this, the **extensibility** of TAS is also important. New data sources like new dictionaries or algorithms, for example optimized translation models or specialized models for term extractions, should be easily incorporated into the system and replace outdated or lower quality methods. I try to account for **usability** and **performance**, but I might have to use algorithms with slower performance that improve the quality of the output, in order to avoid inaccurate or unusable generated definitions.

## 4.3 Subsystem Decomposition

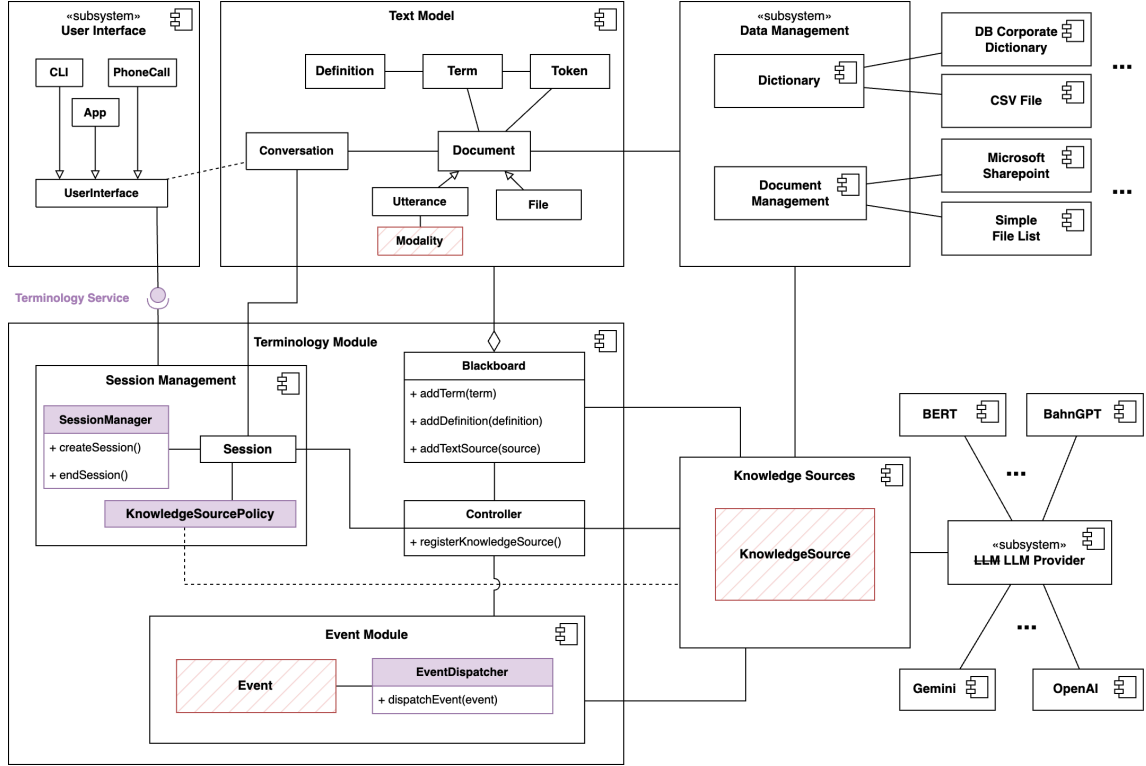


Figure 8: Subsystem decomposition of the Terminology Agent System (UML component diagram).

Figure 8 shows the subsystem decomposition of TAS. New or changed classes or components are marked in **purple**. The system is separated into 4 major subsystems: the User Interface, the Terminology Module, the KnowledgeSources and Data Management.

The User Interface subsystem only holds UI components, the specialized implementations of **Documents**, their **Modality** and the **Conversation**. It is responsible for the interaction with the user and displaying the results of the TAS.

The Terminology Module groups the core logic of the system. The **Request** and **Response** classes are combined in the **Terminology Service**, which provides an API for the **UserInterface**. Using it, a new session can be started (e.g. “analyze this PDF I’m sending you” or “analyze this sentence from a conversation”). The classes in the **TextModel** subsystem, like the Document or Term class, do not

expose public methods themselves anymore, as either some subsystems don't have access to the actual instance (e.g. UI vs. Terminology Module) or the actual algorithm is not placed in one of the TextModels themselves but as seen later in a respective KnowledgeSource. This also removes dependencies between TextModels and actual solution strategies.

The **Session Management** is responsible for setting up and managing multiple TAS sessions, so that multiple users can use the system at once without interfering in the terminology analysis of other users (NFR8) . The **SessionManager** creates and ends **Sessions**. Every **Session** keeps multiple **Conversations** as context (see Figure 3), as well as a **Controller** instance and a newly introduced **KnowledgeSourcePolicy**, which sets the configuration of the TAS for the respective Session. For example, the policy could specify, that only LLMs should be used, therefore the **Session** is initialized only with Knowledge Sources that employ LLMs. The selected Knowledge Sources are therefore known before an event is published. This enables the actual implementations of the Knowledge Sources to be initialized before they may be invoked. For example, a connection to a dictionary database can be established beforehand, since it is known that it will be used to access a dictionary entry. Moreover, existing Knowledge Sources do not have to be changed if new algorithms or data sources are added, only the policy has to be updated. The **Session** therefore acts as Context and the **KnowledgeSourcePolicy** as Policy, similar to the Strategy Pattern [BD10]. The single strategies are then the Knowledge Sources. The Session is then responsible for starting the TAS using the **Controller**. It contains the **Blackboard**, which in turn holds the current state of the **TextModels**.

The logic for the event flow is grouped in the **Event Module**. The new **EventDispatcher** is created to handle incoming and outgoing events and decouples the event flow from the **Controller**, in order to maintain separation of concerns. The available **Events** are shown in Figure 4. The **Controller** registers Knowledge Sources as handlers in the **EventDispatcher**.



The **KnowledgeSources** are grouped in a separate subsystem. The available **KnowledgeSources** are shown in Figure 5. They depend on a **Blackboard** and **Controller** and can optionally use external sources. **KnowledgeSources** are the only components accessing external systems. For example, a **OccurrenceResolver** might use some kind of **Document Storage** to find documents containing a certain term, or a **TermExtractor** might access a LLM running on a different system. KnowledgeSources are independent from each other and can be attached or detached without breaking the system. This allows single algorithms to be tested in isolation. Since the **Blackboard** and **Text Models** only hold state and do not perform any actions, just the data models have to be initialized.

There is no generic abstraction for LLM providers. Not having a single a LLM interface has the drawback that LLM providers cannot be changed easily, as each Knowledge Source that uses LLMs has to be updated to the new model. Even if the way in which LLMs are used is very similar and models can often be interchanged<sup>11</sup>, there are small differences between the vendors and models. For example, the reasoning model from OpenAI like “o4-mini” does not support the `temperature` parameter. As another example, the Gemini models support specifying a schema for the output format [Goo25], which could further optimize or change the quality of the model’s output. Since the interfaces are slightly different but could change the models performance significantly, it would be complicated to design a unified interface for existing and future LLMs (NFR1) .

External data sources or external systems are separated in the **Data Management** subsystem. They are not scope of this thesis. Knowledge sources might access them, but TAS is not reliant on any data management system. Its up to the Knowledge Sources to use external systems.

The TAS can now be used in multiple situations: When LLMs are available or allowed, the policy can be changed to enable or disable the respective Knowledge Sources. When a new client for the TAS is developed, the UI can reuse the

---

<sup>11</sup>For example, Anthropic provides an OpenAI SDK compatibility layer: <https://docs.anthropic.com/en/api/openai-sdk>

**Terminology Service.** Changes to the internals of the TAS do not affect the UI since the **Terminology Service** will stay stable. When new data sources become available, a new Knowledge Source can be implemented that uses this new source, only the policy has to be adjusted.

## 4.4 Hardware / Software Mapping

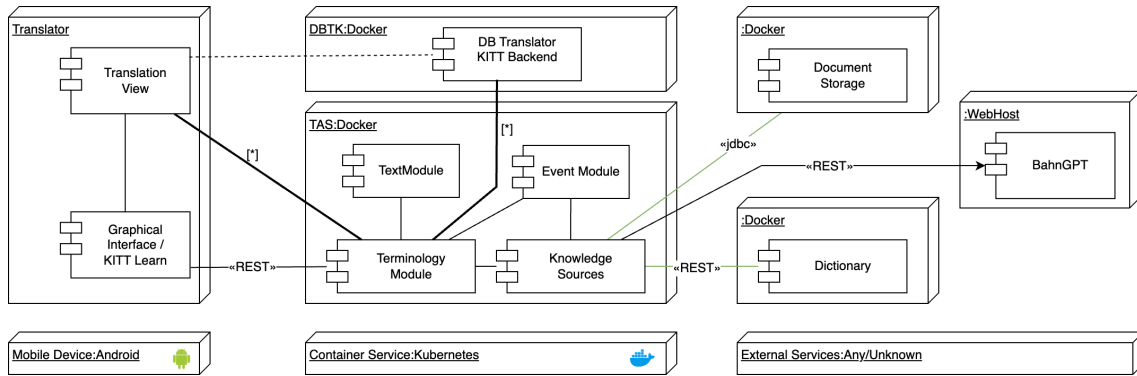


Figure 9: The communication between the subsystems and their deployments (UML deployment diagram).<sup>[\*]</sup>

The **UserInterface** will be implemented as an UI component called **KITT Learn**, which will be integrated into the **DB Translator KITT App**. The new component accesses the TAS as seen in Figure 8 and uses the transcribed text in conversations from the translation screen in DB Translator KITT to display the respective terminology information. However, any other client implementation may use the TAS. The **DB Translator KITT** app is implemented in KOTLIN [Jet24] and uses JETPACK COMPOSE [Goo24] for the UI, which the new module will also use. PYTHON is used as the programming language for the backend, as it is widely used in the machine learning and natural language processing community and provides easy access to pretrained models (e.g. through TRANSFORMERS [Hug24]) and common NLP tools (like SPACY [Exp24]).

<sup>[\*]</sup> currently, the TranslationView directly uses the DB Translator KITT Backend. The invocation of the translation could now be done by the TerminologyModule, the DB Translator KITT Backend would be integrated by a new Knowledge Source, that handles a TranslationRequest event

The **Terminology Module**, the **Knowledge Sources** and its accompanied **Text Module** and **Event Module** are deployed in a single container on a container server at Deutsche Bahn, which decouples it from any other higher demand services. Even though the system is intended to be used in connection with the existing translation screen in the app and the **DB Translator KITT Backend** at Deutsche Bahn, it should not interfere with the development and deployment of those other service. Since the system is intended for use at Deutsche Bahn, DOCKER [Doc24] is used to create containers, which will be deployed on a KUBERNETES [The24] namespace at DB.

The TAS will use LLMs, specifically OpenAI models and BahnGPT, to solve most of the linguistic problems, for example the automatic term extraction. BahnGPT is a wrapper for the OpenAI models and is the only permitted Large Language Model available at Deutsche Bahn. Since it and many other LLMs like Gemini [Dee24], LLaMa [Met24] or Claude [Ant24] have similar APIs and functionality, BahnGPT should be easily replaceable by one of the other models.

I am using pretrained general purpose models, since fine tuning for different domains requires knowledge in machine learning as well as the required hardware, which is often costly and time consuming (NFR4) . The system should easily be used in different domains and thus also provide reasonable results without any modifications to the underlying models.

The association between the **Text Model** and **Document Storage** components has been removed in this model. Since the TAS should exist independently of other components, and the actual algorithms are invoked inside of the **Knowledge Sources**, there is no actual relation e.g. from the **Document** to the **Document Storage**. However, similar data models from the **Text Module** might be used in the **Document Storage** or between the different subsystems.

## 4.5 Continuous Integration and Delivery

For continuous integration, The Terminology Agent System is hosted on a GITLAB [Git24] instance, that automatically runs a pipeline on every commit.

The project is deployed on a KUBERNETES cluster at Deutsche Bahn in a separate DOCKER container. The included *Dockerfile* is tailored to the CI/CD environment at Deutsche Bahn and uses internal base images and configuration to comply with internal security guidelines [DB 25d]. Every push to the main branch initiates a CI/CD pipeline with a build, test and validation stage and other jobs that perform internal compliance and security checks of DB Systel. During the test stage, the *unittests* present in the project will not be performed to prevent potentially expensive tests (e.g. because of the cost of accessing LLMs) to run every commit. The DOCKER container is automatically deployed on the KUBERNETES instance.

## 4.6 Persistent Data Management

The storage of data is organized in the **Data Management**. The registered Knowledge Sources decide whether to use those components and persist data they generate. For example, the **DefinitionGenerator** might save a generated definition in an existing ontology. However, how generated definitions are stored persistently or how added documents are saved will not be discussed in this thesis.

## 4.7 Access Control

Access control is not in the scope of this thesis, as the TAS itself has to be secured depending on the external system. For the product test however, TAS will be deployed at Deutsche Bahn and an authentication proxy is added to the system.

## 4.8 Global Software Control

The global software control of the TAS is partially centralized and partially decentralized.

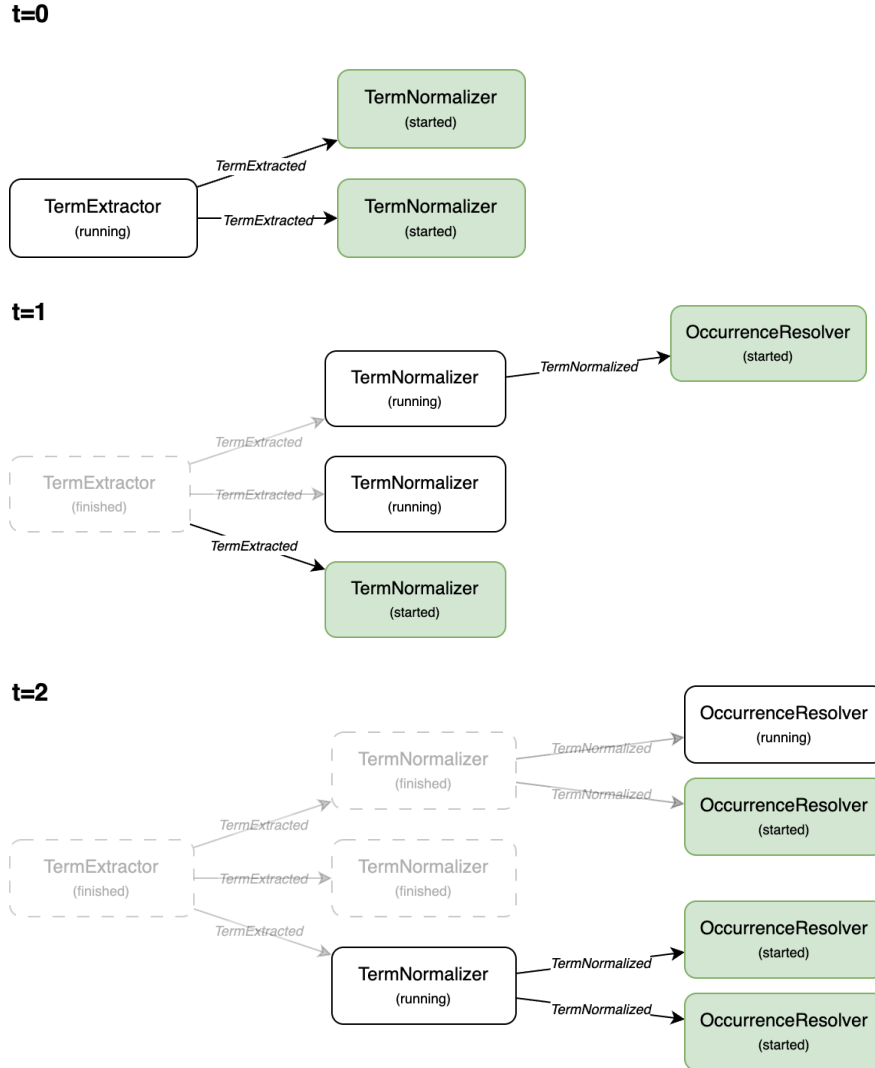


Figure 10: An example event flow in TAS: *White boxes*: running KS, *Green boxes*: starting KS, *Transparent boxes*: finished KS

---

The methods of the Knowledge Sources are invoked through emitting and listening to events. The system is therefore event based. Even though the **Terminology Service**, more specifically the **EventDispatcher**, is responsible for moderating the event flow and thus would pose a centralized point of control flow, the Knowledge Sources themselves actually decide the next steps by emitting and reacting to events. Because of this, the actual control flow is decided by the Knowledge Sources, the **EventDispatcher** solely handles and manages the event flow. This

allows the Knowledge Sources to operate independently and concurrently, since they can asynchronously publish new events and asynchronously listen to them.

Figure 10 gives an example event flow in the TAS. At some point, a TermExtractor started extracting terms from a text.

- (**t=0**) During execution, it emitted the *TermExtracted* event twice, which caused two TermNormalizers to spawn. Those three Knowledge Sources run concurrently.
- (**t=1**) The TermExtractor emitted one last *TermExtracted* event, which launched a new TermNormalizer. At the same time, the upper TermNormalizer normalized a term and emitted the event *TermNormalized*, which in turn started a OccurrenceResolver. The second TermNormalizer is still running. The TermExtractor finished by then.
- (**t=2**) The first TermNormalizer emitted a second TermNormalized event and started a new OccurrenceResolver. Meanwhile, the first and the second TermNormalizer finished, while the third one is still running and emitted two TermNormalized events and started two OccurrenceResolver.

## 5 Object Design

In this section goes into further detail on the design of the individual Knowledge Sources. Section 5.1 annotates two scenarios from Section 3.2 with public methods from the system design. Section 5.2 describes the process of prompt engineering and discusses relevant techniques for this thesis. Section 5.3 explains the design decisions for the prompts used by the Knowledge Sources.

### 5.1 Annotated Scenarios

#### 5.1.1 Understanding instructions

The Hungarian constructor **Gábor** of an ICE 3 train recognizes a defect on the on-board toilet. He looks up the respective instruction book on how to fix the clogged pipe.

```
» SharePoint.queryDocument("ICE 3", "Toilet", "Issue") # External system
```

As he reads the instructions, some terms, for example “Fäkalienbehälter”, are unknown to him, so he uses the DB Translator KITT App to translate the sentence, in which the term occurs.

```
» TerminologyService.translate(transcription, from: "de-DE", to: "hu-HU")
```

The translation alone is not enough to fully comprehend the meaning of the sentence, since he doesn’t know the term “székletgyűjtő tartály”. He taps on the translation to get an overview of contained terms and finds the unknown term.

```
» TerminologyService.processText(forText: transcription)
```

He reads the definition of the term and is now able to fully understand the instructions.

### 5.1.2 Learning “Richtlinien”

Train driver Louise is to be deployed on the route between France and Germany for the next timetable change. She already has the required language qualifications, but needs to revise some of the rules for rail services in Germany. Therefore, she gets the official rulebook Ril 302.6000 [DB 24a] from Deutsche Bahn.

```
» SharePoint.queryDocument("Ril 302") # External system
```

She reads the following sentence: “Streckenfernsprechleitung (Zugmeldeleitung) zwischen Fdl Bantzenheim und Fdl Neuenburg. Der Fdl Bantzenheim kann den Fdl Neuenburg über die Taste „Cantonnement Neuenburg“ erreichen.”. However, she does not fully understand all the words and their definitions.

She opens the DB Translator KITT App on her phone, enters the sentence and taps the “Translate” button.

```
» TerminologyService.translate(text, from: "de-DE", to: "fr-FR")
```

After the translation is shown on the screen, she taps on it and sees the term “Streckenfernsprechleitung” and “Zugmeldeleitung” and the French translation, which is “circuit téléphonique omnibus”.

```
» TerminologyService.processText(forText: text)
```

Underneath the term “Streckenfernsprechleitung”, she reads the official definition and continues her work.

## 5.2 Prompt Engineering and Application of LLMs

*Prompt engineering* is the process of creating and refining instructions - known as prompts - for large language models, in order to guide them in producing responses that contain the desired information and follow a suitable structure for a given task or problem [BZ25].

Building a good prompt is crucial for designing applications that use LLMs. It involves parsing user input, based on that, finding additional context, setting up the prompt for chat completion and then parsing the models output so that it can be used in the application [BZ25]. Since every subtask of TAS is handled by a separate Knowledge Source, each problem in itself has a low level of abstraction. Especially the definition generation task requires a lot of additional context other than the information supplied by the user. Results from the model need to be stored as state and are sometimes required for consecutive prompts, for example the definition generation might reuse previously generated definitions.

There are different techniques for designing a good prompt:

- **Zero-Shot:** The model is only given an instruction and the users context. It has to figure out how to respond by itself. The content of the instruction is very important for this technique compared to others [Sch+25].
- **Few-Shot:** The instruction includes multiple examples of the desired output. Model recognizes patterns in the input and thereby temporarily learns to perform the requested task.

However, prompts can grow very large: if examples themselves are already big (e.g. they contain many attributes), already a few examples can increase the prompt size and maybe even exceed the models' context window [BZ25]. [Sch+25] suggests that quality improvements diminish at around 20 examples. Another drawback is a potential bias towards the examples: examples set expectations to the model and they might influence the models output. Including edge cases can give the model a hint on how to handle non-trivial input [BZ25].

- **Chain-Of-Thought:** Encourage the model to “think” out loud and elaborate on its solution path, which likely improves the correctness of the output



[Wei+23]. Figure 11 shows an example for such a prompt. Instead of directly returning the answer, the given example first elaborates on the question, splits the problem into intermediate steps and argues based on interim result, before it makes a conclusion.

This technique prevents the model to argue on an initial guess instead of answering based on the models’ internal knowledge. It showed significant improvements in arithmetic, symbolic and commonsense reasoning [Wei+23].

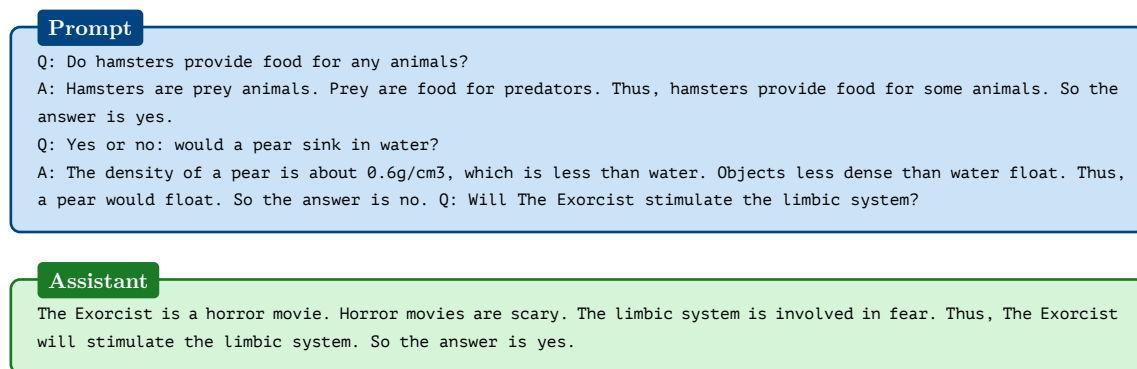


Figure 11: An example for Chain-Of-Thought prompting and a possible result from a model, from [BZ25]

- **Prompt-Chaining:** *Prompt Chaining* divides a task into smaller subtasks that are handled by the model separately to increase accuracy [Ant25]. Compared to *Stepwise Prompt* [Ope25] each subtask is processed as its own prompt instead of having a larger prompt containing all tasks. This approach can prevent models from omitting steps and can simplify debugging of subtasks.

### 5.3 Knowledge Sources

The Knowledge Sources that employ LLMs to process text and solve NLP tasks, all use OpenAI’s “gpt-4o-mini” model to reduce cost. In general, switching to a higher-quality model or a different should be possible, but is not tested in this thesis.

As [Ren24] and [Pat+24] suggest, changing the models temperature<sup>13</sup> does not impact its performance significantly. To increase consistent output, the temperature is thus set to zero. Therefore, the top\_p<sup>14</sup> value has no impact on the output, since only the most probable token is chosen. To increase the chance for deterministic output, the `seed` parameter offered by the OpenAI's API is used.

In the following sections, the prompts used by the Knowledge Sources will be shown. The templates contain placeholders surrounded by double angle brackets (e.g. <<input>>), which are intended to be replaced with the variable input values.

### 5.3.1 TextExtractor

The Pdf2Text text extractor activates with the `DocumentAdded` event. This implementation should support processing PDF files. First, PYPDF[Mat+24] is used to split the document into single pages and then each page is processed using DOCLING[Aue+25] a separate thread to extract structured markdown text. Once a thread finishes extraction, a `TextExtracted` event is fired with the extracted text.

### 5.3.2 TermExtractor

#### Developer

Du bist Experte für Terminologie und Fachbegriffe.  
Deine Aufgabe besteht darin, aus einem Text Begriffe, Abkürzungen und Phrasen zu extrahieren.  
Du extrahierst nur Terminologie, die wahrscheinlich in der Eisenbahn verwendet wird.  
Du erkennst Abkürzungen und behältst sie unverändert bei. Nur wenn die vollständige Form vorhanden ist, fügst du sie in Klammern am Ende des Begriffs an.  
Du extrahierst Phrasen und Wörter sowie verschachtelte Begriffe und deren Einzelteile.  
Achte bei längeren Phrasen darauf, ob aus dem Text klar wird, dass es sich um einen besonderen Begriff handelt, der wahrscheinlich verwendet wird.  
Beginne mit den Begriffen, die am wahrscheinlichsten relevant sind. Gib nur eine Liste von Begriffen zurück.  
Extrahiere nur Begriffe, die besonders für den Kontext "Eisenbahn" sind!

#### User

Input:  
Du musst das Hauptsignal auf Fahrt stellen.

---

<sup>13</sup>Temperature controls the randomness of the output [DAI24]. Higher values increase the likelihood of unlikely tokens.

<sup>14</sup>Top-P sampling is a method to reduce the number of considered tokens by truncating very unlikely token before sampling [Hol+20]. High top\_p values result in more diverse output.

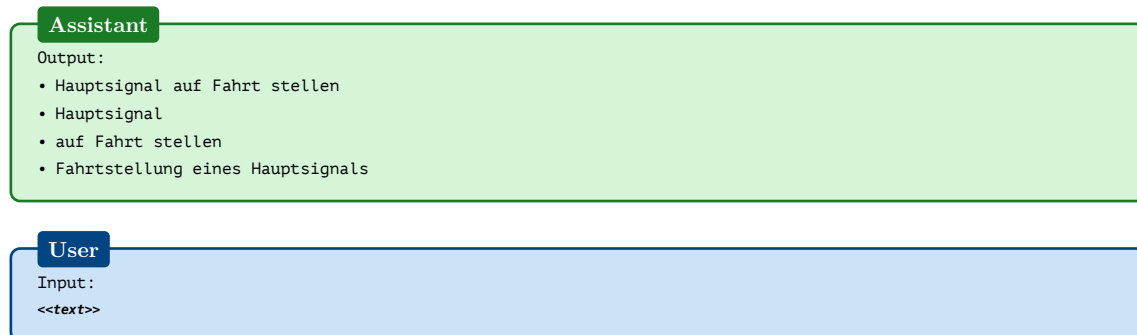
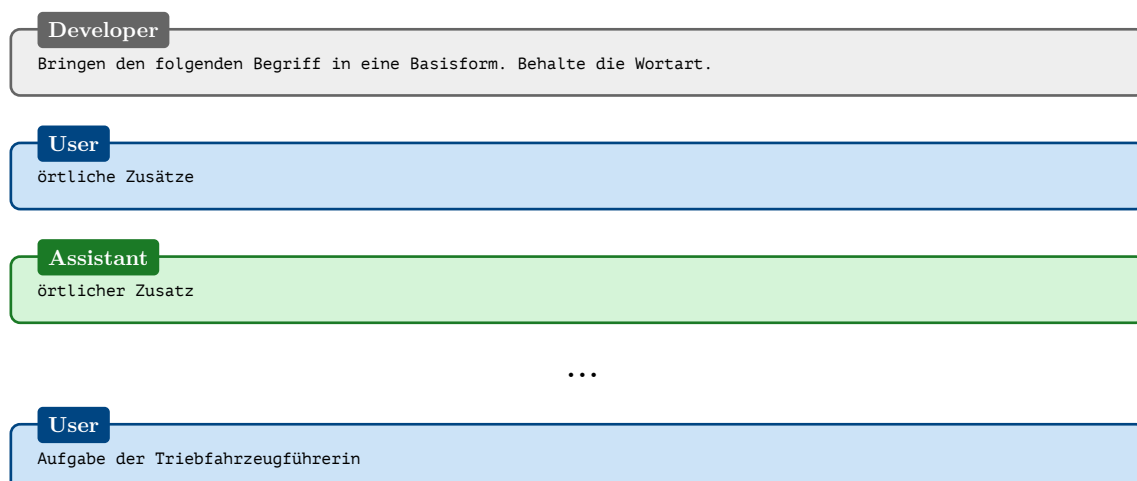


Figure 12: The prompt used for term extraction.

The `TermExtractor` listens to the `TextExtracted` event. The `OpenAITermExtractor` is a subclass of `TermExtractor` and uses OpenAI's “gpt-4o-model” to extract terminology from a text. Figure 12 shows the prompt used for term extraction. The LLM is instructed to extract only terminology from a text that belongs to the railway domain. Phrases, words and abbreviations should be extracted if the full form of an abbreviation is also available, it should be added to the term in brackets. Phrases should only be extracted if they are specific to the domain. After the developer prompt, a one-shot example is given to the model to show the expected output format. At the end, the input is given as shown in Figure 12. For each extracted term, a `TermExtracted` and `OccurrenceResolved` event is published.

### 5.3.3 TermNormalizer



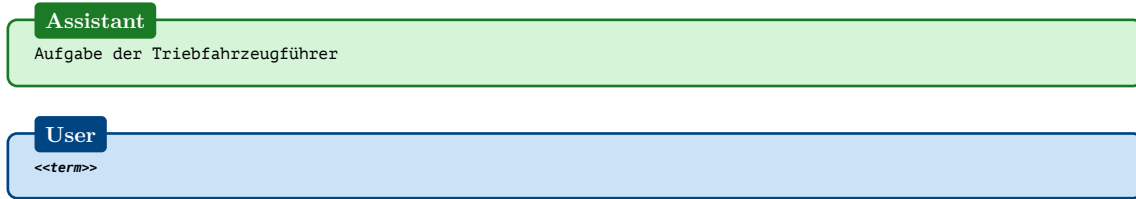


Figure 13: The prompt used for term normalization.

The **TermNormalizer** activates on the **TermExtracted** event. Only the LLM based **OpenAITermNormalizer** is implemented. The problem is solved using a few-shot prompt, by giving an initial instruction and several examples with different cases of text normalization, such as transforming terms to singular, neuter or gender neutral forms. After the response, the normalized term is stored on the **Blackboard** and a **TermNormalized** event is published.

#### 5.3.4 DefinitionGenerator and DefinitionCombiner

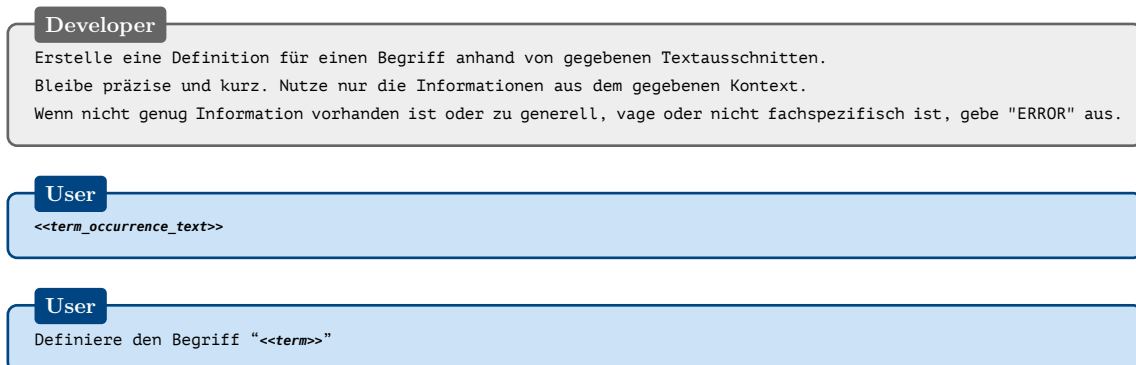


Figure 14: The prompt used for definition generation.

In the concrete object design, the LLM based **DefinitionResolver** will be split into two separate Knowledge Sources. The **OpenAIDefinitionGenerator** listens to the **OccurrenceResolved** event. For every new text occurrence of a term, a new definition based on that text section is generated. If the term can be matched in the text string, only the term and the context around that term (200 characters before and 300 characters after the term) is used. This should reduce the number of tokens passed to the LLM and remove potentially unnecessary information from the definition generation process. If the term is matched more than once, a defin-

ition is generated for each match. The GPT model is then instructed to generate a definition for the term based on the context (see Figure 14). If there is not enough information in the text to properly define the term, it should stop completion with the “ERROR” string, in which case no definition is added to the term. If the “ERROR” token was considered as the first token with a probability of  $p_{\text{ERROR}} \geq 0.05$ , the definition is discarded, even if the model returned a complete text. Each valid generated definition publishes an **PartialDefinitionGenerated** event. It is considered *partial*, as it may not reflect the ground truth meaning of the term. For example, if the term occurred casually in a sentence (e.g. in an enumeration of tasks), the surrounding context may distort the generated definition.

The **OpenAIDefinitionCombiner** listens to the **PartialDefinitionGenerated** event and is responsible for generating a combined definition for a term. If a term has at least three partial definitions, each definition is classified by an LLM, whether the generated text is specific enough and likely defines the respective term (see Figure 29).

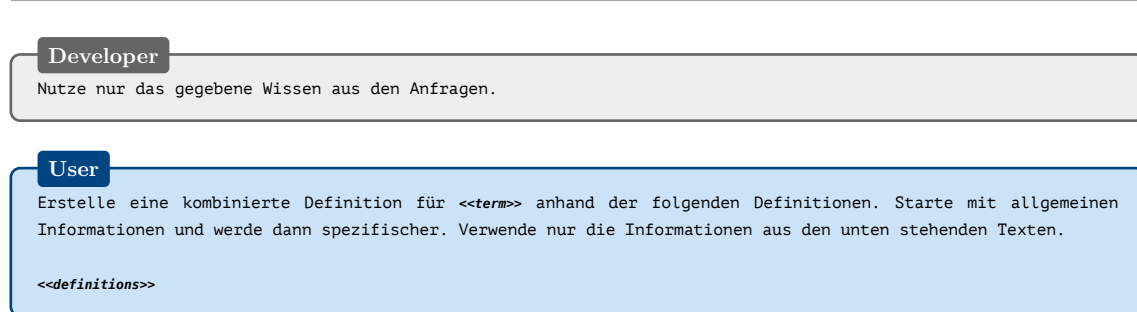


Figure 15: The prompt used for combining relevant definitions.

If the model returns “TRUE”, it is added as context for the definition combiner. The model is instructed to generate a combined definition only based on the texts given in the prompt (see Figure 15). It is important that only the generated text is considered to avoid bias of the model influencing the generated definition. The combined definition is then added to the term and published as a **CombinedDefinitionGenerated** event.

### 5.3.5 CSVDefinitionResolver

As an additional source, the `CSVDefinitionResolver` looks for verified definitions in a CSV file. It listens to the `TermExtracted` and `TermNormalized` events. The CSV file contains terms and definitions extracted from [Thi21].

## 6 Extensibility of TAS

A key benefit of TAS is its extensibility and adaptability (NFR1) . This design allows for the seamless integration of new data sources or algorithms as they become available. This chapter illustrates the methodology by which the TAS can be augmented with additional Knowledge Sources, demonstrating its capacity for expansion and enhancement.

First in Section 6.1, the **TermExtractor** and **TermNormalizer** are adapted to use the internal GPT model at Deutsche Bahn. Then, Section 6.2 introduces an implementation of a new algorithm for term extraction. Finally, a new **OccurrenceResolver** is introduced in Section 6.3.

---

```
Python
1  # 1. Define a new subclass of KnowledgeSource
2  class MyKnowledgeSource(KnowledgeSource):
3      # 2. Define which events this KnowledgeSource listens to
4      handles: List[Type[Event]] = [TermExtracted, OccurrenceResolved]
5
6      # 3. Implement the algorithm
7      async def activate(self, event: Event) -> AsyncIterable[Event]:
8          for i in range(10):
9              # 4. Yield events once they are ready to be processed
10             yield TermNormalized()
```

---

Figure 16: Template code for adding a new Knowledge Source

---

In order to create a new Knowledge Source, a new class inheriting from `KnowledgeSource` (or a specialized subclass of `KnowledgeSource`) has to be implemented. Each Knowledge Source consists of a list of listened events and an `async def activate()` function that implements the actual logic of the Knowledge Source. The activation function has to return an `AsyncIterable` (or at least call `yield`

once) of type `Event`. Every time the Knowledge Source generates a new event (for example new results are available), a new `Event` has to be yielded.

## 6.1 AI models - BahnGPT

In the case of Deutsche Bahn, the use of Generative AI is generally prohibited or subject to high requirements and strict regulations [Deu24b]. In order to comply with the regulations at Deutsche Bahn and still enable the use of Large Language Models, DB Systel introduced BahnGPT, an interface that provides a version of a GPT4 model that conforms to the guidelines of the company [DB 25c].

For TAS to be deployed at Deutsche Bahn, the `OpenAITermExtractor` cannot be used anymore. Therefore, the invocation of the LLM has been abstracted and a new superclass `LLMTermExtractor` was created, since the models from BahnGPT and OpenAI are identical in usage. The `LLMTermExtractor` class now only contains the raw prompts and the logic relevant for pre- and postprocessing, whereas the `OpenAITermExtractor` now only implements the OpenAI model invocation. A new implementation of the `LLMTermExtractor`, called `BahnGPTTermExtractor`, is created, which invokes the BahnGPT API instead of OpenAI.

---

knowledge/bahngpt/extract.py

Python

```
1 class BahnGPTTermExtractor(LLMTermExtractor):
2     async def get_llm_response(self, text: str) -> str:
3         return await create_completion_bahngpt(
4             messages=[
5                 ("developer", f"{DEVELOPER_PROMPT}"),
6                 ("user", EXAMPLE_USER),
7                 ("assistant", OUTPUT_ASSISTANT),
8                 ("user", "Input: \n" + text)
9             ]
10         )
```

---

Figure 17: Implementation of new Knowledge Source `BahnGPTTermExtractor`

A similar process is also required for the `OpenAILEmmatizer`, `OpenAIDefinitionGenerator` and `OpenAIDefinitionCombiner`. All of the logic other than the LLM invocation of these Knowledge Sources has been abstracted.

For the new Knowledge Source to be available in the TAS, the `Session` has to register it on the `Session's Controller` object.

```
session.py Python
27 class Session(BaseModel):
28     id: Annotated[UUID, Field(default_factory=uuid.uuid4)]
29     policy: KnowledgeSourcePolicy
...
35 def setup_controller_term_extraction(self, controller: Controller):
36     controller.register_knowledge_source(CSVDefinitionResolver)
37     if self.policy.use_llm:
38         controller.register_knowledge_source(OpenAIExtractor)
39         controller.register_knowledge_source(OpenAILemmatizer)
40 +     if self.policy.use_bahngpt:
41 +         controller.register_knowledge_source(BahnGPTTermExtractor)
42 +         controller.register_knowledge_source(BahnGPTTermLemmatizer)
43
44     return controller
45
46 def setup_controller_definition_generation(self, controller: Controller) :
47     if self.policy.use_llm:
48         controller.register_knowledge_source(OpenAIDefinitionGenerator)
49         controller.register_knowledge_source(OpenAIDefinitionCombiner)
50 +     if self.policy.use_bahngpt:
51 +         controller.register_knowledge_source(BahnGPTDefinitionGenerator)
52 +         controller.register_knowledge_source(BahnGPTDefinitionCombiner)
53     return controller
```

Figure 18: Code snippet showing the addition of the newly created Knowledge Sources.

## 6.2 Implementation of new algorithm

To reduce cost resulting from using LLMs, a traditional ATE algorithm, the C-Value [FAM00] should be implemented. The `TermExtractor` base class can be reused to assign the correct event handles. This class uses the SPACY [Exp24] library and uses the C-Value implementation from PYATE [Lu21]. On activation, the extracted text is passed to PYATE and the recognized terms are added to the blackboard as well as published as `OccurrenceResolved` and `TermExtracted` events.



---

**cvalue.py**
 Python

```

1  class CValue(TermExtractor):
2      nlp: Language = None
3
4      def model_post_init(self, __context: Any) -> None:
5          # Initialize the Language model
6          self.nlp = spacy.load("de_core_news_md")
7          lazy_module("pyate").TermExtraction.configure({
8              "language": "de",
9              "model_name": "de_core_news_md",
10             "MAX_WORD_LENGTH": 5
11         })
12
13     async def activate(self, event: TextExtracted) -> AsyncIterable[Event]:
14         candidates = lazy_module("pyate").cvalues(event.text,
15             have_single_word=True).to_dict().keys()
16         source = self.blackboard.add_text_source(event.text)
17         for term in candidates:
18             t = self.blackboard.add_term(term)
19             yield OccurrenceResolved(term=t, source=source)
20             yield TermExtracted(term=t)

```

---

Figure 19: The implementation of a new Knowledge Source employing the C-Value algorithm.

---

## 6.3 Implementation of new data source

To increase the quality of the definition generation, relevant documents with context to a term should be incorporated into our process. At Deutsche Bahn, the “Konzernregelwerksdatenbank” (short KRWD) [Deu25d] stores all available guidelines and policies at DB and makes it possible to query relevant documents by keywords. This data source can easily be integrated into TAS.

---

**krwd.py**
 Python

```

1  class KRWD:
2      async def query(self, keywords: list[str]) -> list[str]:
3          # Returns a list of strings, representing the documents containing the passed keywords
4          pass

```

---

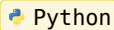
Figure 20: Code snippet showing the interface for the KRWD.

---

Since the bureaucratic effort was too high to get access to this system, the interface of the KRWD is mocked. Its only method is to query documents containing certain keywords.

This interface is used in the new Knowledge Source `KRWD0ccurrenceResolver` which inherits from the already specialized `OccurrenceResolver`. The `OccurrenceResolver` listens to the `TermExtracted` and `TermNormalized` events. The Knowledge Source queries the KRWD for documents containing the term of the input event and emits the `OccurrenceResolved` event for every document.

---

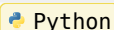
```
krwd_occurrence.py  Python
1 class KRWD0ccurrenceResolver(OccurrenceResolver):
2     krwd: KRWD
3
4     async def activate(self, event: TermExtracted | TermNormalized) -> AsyncIterable[Event]:
5         term = event.term
6         result = await krwd.query([event.term.normalized_or_text()])
7
8         for document in result:
9             source = self.blackboard.add_text_source(document)
10            term.occurrences.append(source.id)
11            yield OccurrenceResolved(term=term, source=source)
```

---

Figure 21: The implementation of the newly introduced `KRWD0ccurrenceResolver`.

Again, this Knowledge Source has to be added to the `Session` and registered to the `Controller` object:

---

```
session.py  Python
36 def setup_controller_definition_generation(self, controller: Controller):
...
40 + controller.register_knowledge_source(KRWD0ccurrenceResolver)
41 return controller
```

---

Figure 22: Add the Knowledge Source in the `SessionManager`

The same process can be applied to the introduction of any new algorithm, model, data source or functionality. Technically, also new interfaces for data export or import can be implemented: a state exporter could be attached that listens to

all events and permanently stores the state of the blackboard or controller in a database, or a data importer could be added that loads previously saved context or session data. Since each Knowledge Source runs independently, the functionality of other modules and Knowledge Sources is prevented.

## 7 Testing

This chapter discusses the testing of TAS. Section 7.1 describes two unit tests testing single Knowledge Sources. Section 7.2 integrates multiple components of TAS and tests them in combination. Section 7.4 discusses problems that arose when testing applications that employ LLMs. Section 7.3 presents a system test that was run at Deutsche Bahn and shows the performance of the term extraction and definition generation. Section 7.5 details threads to the validity of the previous tests.

When working with generative models, particularly large language models (LLMs), it is essential to establish a robust testing framework to ensure high-quality output and the safe use of the software. One key challenge arises from the inherently non-deterministic nature of LLM outputs. As demonstrated by [Ati+25], even when using deterministic settings, significant variations can still occur between different runs. This non-determinism has several practical implications: it can lead to unusable outputs due to formatting issues, produce non-reproducible and potentially incorrect information, and complicate unit testing. In particular, repeated executions of the same prompt may yield different results, which could mistakenly be interpreted as regressions. Furthermore, even slight variations in output—despite consistent settings—can interfere with automated testing procedures, as will be illustrated in later sections.

### 7.1 Unit Test

An advantage of the TAS architecture is that each Knowledge Source can be executed and tested independently. The LLM-based implementations of the `OpenAITermExtractor` and `OpenAIDefinitionGenerator` will undergo testing. As test

data, selected excerpts from the “Fahrdienstvorschrift 40820” [DB 24b] are used, as this document contains verified domain-specific terminology from Deutsche Bahn.

<i>Test case name</i>	TestTermExtractor_Common
<i>Participating objects</i>	OpenAITermExtractor
<i>Entry conditions</i>	<p>A <code>TextExtracted</code> event is passed to the <code>OpenAITermExtractor</code> with the following text:</p> <p><i>“Einseitig gerichtete Sprechereinrichtung verwenden</i>  <i>Aufträge dürfen über einseitig gerichtete Sprechereinrichtungen gegeben werden, wenn dies im Einzelfall nicht verboten ist und der Empfänger die Ausführung melden muss oder der Auftraggeber die Ausführung selbst erkennen kann. Meldungen dürfen über einseitig gerichtete Sprechereinrichtungen nicht gegeben werden.”</i></p>
<i>Expected behaviour</i>	<ul style="list-style-type: none"> <li>• At least the following terms are present on the blackboard: <ul style="list-style-type: none"> <li>▸ “Einseitig gerichtete Sprechereinrichtung”</li> <li>▸ “Aufträge”</li> <li>▸ “Empfänger”</li> <li>▸ “Auftraggeber”</li> <li>▸ “Meldungen”</li> </ul> </li> <li>• The following events have been published: <ul style="list-style-type: none"> <li>▸ for each term: <code>TermExtracted(term)</code></li> <li>▸ for each term: <code>OccurrenceResolved(term, source_text)</code></li> </ul> </li> </ul>

Figure 23: Test description for TestTermExtractor\_Common

Figure 23 shows the documentation of a common test case for the `OpenAITermExtractor`. A portion of a text is passed to the activation function of the `OpenAITermExtractor`. The KS must at least extract a predefined set of terms, otherwise it is considered to have failed. For some terms, variations are allowed, as the `OpenAITermExtractor` is instructed to already normalize terms if possible. For each of the terms, a respective `TermExtracted` and `OccurrenceResolved` event must be published.

<i>Test case name</i>	TestDefinitionGenerator_Simple
<i>Participating objects</i>	OpenAIDefinitionGenerator
<i>Entry conditions</i>	<p><code>OccurrenceResolved</code> event is passed with the following parameters:</p> <ul style="list-style-type: none"> <li>• term: “Abstoßen”</li> </ul>

	<ul style="list-style-type: none"> <li>source:</li> </ul> <p>“Abstellen</p> <p>Züge und Triebfahrzeuge sind abgestellt, wenn sie nicht mit einem Triebfahrzeugführer besetzt sind oder nicht gesteuert werden. Wagen sind abgestellt, sofern sie nicht in Züge eingestellt sind oder nicht rangiert werden.</p> <p>Abstoßen</p> <p>Abstoßen ist das Bewegen geschobener, nicht mit einem arbeitenden Triebfahrzeug gekuppelter Fahrzeuge durch Beschleunigen, sodass die Fahrzeuge allein weiterfahren, nachdem das Triebfahrzeug angehalten hat.”</p>
<i>Expected behaviour</i>	<ul style="list-style-type: none"> <li>A new definition is added to the term: “Abstoßen ist das Bewegen geschobener, nicht mit einem arbeitenden Triebfahrzeug gekuppelter Fahrzeuge durch Beschleunigen, sodass die Fahrzeuge allein weiterfahren, nachdem das Triebfahrzeug angehalten hat”</li> <li>The <code>PartialDefinitionGenerated</code> event has been published</li> </ul>

Figure 24: Test description for `TestDefinitionGenerator_Simple`

The test for the `OpenAIDefinitionGenerator` as described in Figure 24 passes the term “Abstoßen” and a portion of a text, containing that term and its exact definition as well as a different term with its exact definition. After execution, the exact definition for “Abstoßen” from the text is expected to be extracted and published as a `PartialDefinitionGenerated` event.

In the special case, that a term and a text is processed by the `OpenAIDefinitionGenerator` and the text does not contain enough information for the term to be defined properly (e.g. the term is only mentioned in passing), no definition should be added and thus no event published. Figure 30 documents a test that passes such parameters and expects the `OpenAIDefinitionGenerator` to not publish any generated definitions.

## 7.2 Integration Test

As seen in Figure 25, TAS follows a layered architecture. At the top, the User Interface represents the user-facing layer, which interacts with the Terminology Service responsible for session management and event handling. This service coordinates the execution of various Knowledge Sources (KS), which in turn access

and process information from underlying external data components. There are different approaches to integration testing: horizontal testing assumes a layered architecture of the components and tests groups of components per layer. The layer under test is attached to the next consecutive layer and tested in combination.

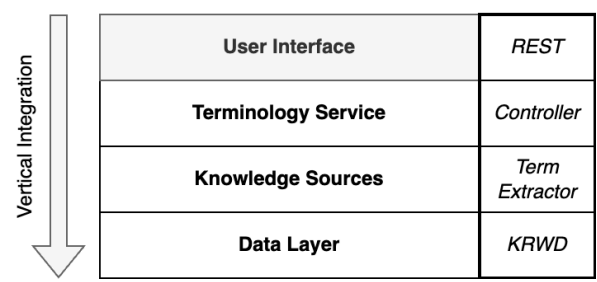


Figure 25: Layered architecture of TAS showing vertical integration from the user interface to the data layer. Each layer encapsulates a specific role in the system

Vertical integration testing takes a different approach by integrating and testing a complete scenario. This requires the identification of all participating components and the integration between them to produce an integration test. A combination of both hierarchical and vertical integration is used to test parts of the system. The test case documented in Figure 26 integrates the use case in Figure 2. For this, REST API, the SessionManager, the EventDispatcher and the Knowledge Sources TermExtractor and TermNormalizer are required. The actual user interface (the Android App) will not be tested to keep the scope of this thesis. Instead, the Terminology Service and its REST API will act as the top most layer.

<i>Test case name</i>	ExtractDomainTerminology_Test
<i>Participating objects</i>	<ul style="list-style-type: none"> <li>• TerminologyService (<i>REST API</i>)</li> <li>• SessionManager</li> <li>• EventDispatcher</li> <li>• OpenAITermExtractor</li> <li>• OpenAITermNormalizer</li> </ul>
<i>Entry conditions</i>	<ul style="list-style-type: none"> <li>• The following utterances have been made: <ul style="list-style-type: none"> <li>▸ User A: “Servus Zofia!”</li> <li>▸ User B: “Hallo Markus.”</li> </ul> </li> </ul>

	<ul style="list-style-type: none"> <li>► User A: “Rangiere mir bitte mal den 420er von Gleis 3 auf das Abstellgleis. Passt auf, du musst auf Sicht bis zu den Signalen fahren.”</li> </ul>
<i>Flow of events</i>	<ol style="list-style-type: none"> <li>1. A <code>HTTP Request</code> to <code>/extractTerminology</code> for extracting terminology is sent to the <code>TerminologyService</code> including the last <code>Utterance</code> and the other <code>Utterances</code> as context.</li> <li>2. The <code>TextExtracted</code> event is published to the <code>Blackboard</code>.</li> <li>3. The <code>OpenAITermExtractor</code> activates and extracts at least the following terms: “Rangieren”, “420”, “Gleis”, “Abstellgleis”, “auf Sicht”, “Fahren auf Sicht”, “Signale”. It creates a <code>Term</code> object for each of them and stores them on the blackboard.</li> <li>4. For each term a new <code>TermExtracted</code> event is published.</li> <li>5. The <code>OpenAITermNormalizer</code> activates and normalizes the term “Signale” to “Signal”. It publishes a <code>TermNormalized</code> event and updates the <code>Term</code> object.</li> <li>6. The TAS halts and returns a <code>TerminologyResponse</code> containing all the terms that were added to the blackboard.</li> </ol>
<i>Expected behaviour</i>	<ul style="list-style-type: none"> <li>• The following terms should be included in the response: “Rangieren”, “420”, “Gleis”, “Abstellgleis”, “auf Sicht fahren”, “Signal”</li> </ul>

Figure 26: Test description for `ExtractDomainTerminology_Test`

The integration test in Figure 26 receives an initial HTTP request to start term extraction. During execution, the `OpenAITermExtractor` and `OpenAITermNormalizer` should be activated. The test is successful, when the response contains all of the expected terms.

Performing the test presented a weakness in the current testing method. Even though the LLM response was parsed and terms were extracted successfully, the actual terms slightly differed from the expected behavior: Instead of “420”, the term “420er”<sup>15</sup> was extracted (see Appendix B). In a different imaginary test case, the `TermExtractor` could have returned “Fahrt auf Sicht”, whereas “auf Sicht fahren” would have been expected. In the classical interpretation, this would be a test failure, however, in this case the result is not wrong and must be validated manually. This situation opens the possibility to learn from the unexpected output and extend the expected behavior to allow multiple variants.

<sup>15</sup>“420” is the official model designation of a train class [Wik], whereas “420er” refers to an instance of this model

For small test sets, manual validation is acceptable, however, for large quantities, human verification is time-consuming. There are various approaches to automatically check the similarity of the results:

1. **Fuzzy string matching:** assumes that the differences between terms only lie in some characters missing or added. Find a threshold  $\delta$ , in which two terms are considered similar. The test is successful, if there is an observed term for every expected term within the specified  $\delta$ . For this method to be successful a suitable and universal  $\delta$  must be found. One algorithm for this is the Levenshtein Distance [Lev66]. For example, the terms “Gleis” and “Gleis 3” may be incorrectly considered similar, even though “Gleis 3” should not be treated as a valid term.
2. **Sentence Embedding:** embed every term into a vector representation using a pre-trained model specialized in sentence embedding and compare the vectors pairwise, for example by calculating the cosine similarity between two vectors [Mon+25]. Find a threshold  $\delta$ , in which two terms are considered similar. The success of the test is equal to the previous method. The challenge again lies in determining a suitable  $\delta$ , that matches our interpretation of semantic difference.
3. **Assessment using LLMs:** use prompt engineering to compare the expected outcome with the actual result of the test. [Hus+23] uses a Chain-of-Thought prompt to evaluate the similarity using a score between 0 and 4. A simpler approach is to classify as similar (“TRUE”) or not similar (“FALSE”). Find a probability threshold  $p_{\min_{\text{TRUE}}}$  required for the model to output “TRUE”, so that the test can be considered successful. This method allows to specify criteria for evaluation in natural language.

For this integration test, the method based on LLMs was chosen, as it is the simplest yet most powerful among those previously presented. The prompt used can be seen in Figure 31 (Appendix A). It is important to note, that the model is specifically instructed to reduce output tokens and directly answer with “TRUE” or “FALSE”. When the model is allowed to generate reasoning steps, the probabilities for the final token are largely based on the previously generated



tokens of the reasoning process. As a result, analyzing these probabilities may offer little added value, since the model’s decision regarding the final outcome is often implicitly made during the reasoning itself. For example, if the model argues that the observed results align with the expected results, the probability of the final token being “FALSE” becomes highly unlikely.

### 7.3 System Test

To assess the overall quality of the output of TAS, a performance test (NFR5) (NFR6) was conducted in cooperation with members of the DB Systel team. The evaluation of TAS system was performed manually and utilized the following metrics:

- **Term Recall (Strict) (NFR5)** : This metric quantifies the proportion of expected terms that were retrieved with an exact match. It serves as a measure of the system’s ability to identify predefined terminology without deviation.
- **Term Recall (Lenient) (NFR5)** : In contrast to the strict variant, this metric also accounts for semantically similar terms that, while not identical to the expected ones, have equivalent meaning (see Section 7.4).
- **Term Precision (NFR5)** : This metric evaluates the proportion of extracted terms that are actually relevant. In addition to the predefined terms, it also includes terms that were not originally expected but are considered semantically or contextually appropriate within the railway domain.
- **Validity of Definitions (NFR6)** : This metric assesses the proportion of generated definitions deemed valid in the context of practical application. A definition was considered valid if it would not lead to misunderstandings, operational problems, or safety risks in a real-world working environment.

A limited set of 21 crafted test sentences were created incorporating both colloquial and formal language, as well as terminology specific to Deutsche Bahn. The `OpenAITermExtractor`, the `OpenAITermNormalizer`, the `OpenAIDefinitionGenerator` and `OpenAIDefinitionCombiner` were activated for the test. The test cases were

executed consecutively. The results for the individual test sentences are presented in Appendix A.

The test yielded the following results: The system was able to retrieve 91% of the expected terms with an exact match. When allowing for semantically similar terms, the recall increased to 98%, indicating that the system can recognize almost all of the relevant concepts. Of all the terms extracted, 78% were judged to be relevant to the domain. 84% of the generated definitions were evaluated as valid according to domain-specific criteria, meaning they would not likely cause misunderstandings or safety issues in professional use.

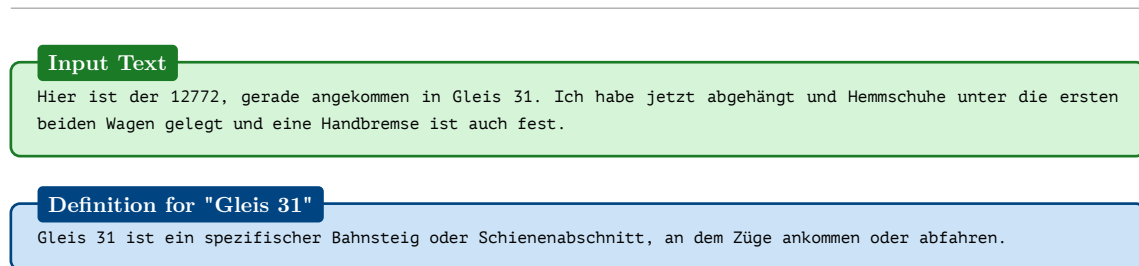


Figure 27: Excerpt from the system tests. The generated definition has little informational value.

---

However, in many cases, TAS generated definitions with low informational content or included information that was not solely derived from the context text. Even though they were not considered inaccurate or could potentially harm workers, those results are not desired. For example, the generated definition for “Gleis 31” in the test case shown in Figure 27 not only has little informational value compared to the input text, but also contains knowledge that was not included in the prompt. This phenomenon falls under what’s known as *hallucination* [Hua+25] - more specifically, data-based hallucination, where the model likely inferred missing context from its training data. Hallucination caused by LLMs is still an active area of research and can’t yet be fully avoided. It refers to cases where LLMs generate output that is false, nonsensical, or incorrect based on the given input context.

Due to the limited number of test cases, it is important to note that the evaluation is not sufficient to draw definitive conclusions about the overall perfor-

mance or reliability of the system. However, the test provides an initial indication of the system’s functionality and highlights its potential for further development.

## 7.4 Difficulties in testing applications using LLMs

The protocol of the integration tests (see Appendix A) illustrates difficulties that arose when testing applications that use non-deterministic generative models like LLMs.

ExtractDomainTerminology_Test		js
1	// probabilities in iteration 1	
2	{'FALSE': 0.6224593312018545, 'TRUE': 0.3775406687981454}	
3	// probabilities in iteration 2	
4	{'FALSE': 0.9149009474519222, 'TRUE': 0.08509905254807776}	
5	// probabilities in iteration 3	
6	{'TRUE': 0.679178699175393, 'FALSE': 0.32082130082460697}	
7	// probabilities in iteration 4	
8	{'FALSE': 0.9399133527714579, 'TRUE': 0.060086647228542005}	
9	// probabilities in iteration 5	
10	{'TRUE': 0.9241418131886822, 'FALSE': 0.0758581868113179}	

Figure 28: Excerpt from test results of ExtractDomainTerminology\_Test. Every iteration resulted in different probabilities for the result token.

Due to the non-deterministic nature of LLMs, it cannot always be guaranteed, that running inference on an LLM with the same input parameters results in the same output. Even though adjusting hyperparameters of LLMs like temperature or top\_p can decrease the randomness of the output, the results were not perfectly consistent. After running the test in Figure 26 five times without changing the input and hyperparameters, the probabilities for the result token (either “TRUE” or “FALSE”) were not consistent. [Ati+25] also obtains similar results: under deterministic settings for LLMs, the LLM also generated inconsistent or non-deterministic output.

Traditional binary classification of test success and failure may not always be applicable when working with language. This is because proposed correct results from experts can be subjective or inconsistent, often due to differences in subdo-

mains. The criteria for determining sentence similarity may vary between experts and subdomains, with, for instance, legal experts typically preferring precise definitions, whereas engineers may tolerate looser formulations. When designing prompts for the LLMs, these criteria must be clearly and accurately expressed to enable the model to draw correct conclusions.

Some of these issues can be resolved by allowing the LLM to reason and express the internal thought process. When running the same tests again without restricting the model to respond with a single token, the reasoning steps and produced tokens were still different, but the final output, whether the list of terms was similar, was correctly and consistently answered (see Appendix A). Reasoning allowed the model to reflect on the given results and provide answers with a high degree of certainty.

## 7.5 Threats to validity

This section addresses the potential threats to the validity of the testing conducted for the Terminology Acquisition System (TAS), categorized into internal, construct, and external validity.

- **Internal Validity.** The inherently non-deterministic nature of LLM outputs poses a significant threat to internal validity.

Repeated executions of the same prompt may yield different results, which could mistakenly be interpreted as regressions, thus influencing the interpretation of test outcomes. Even with consistent settings, slight variations in output can interfere with automated testing procedures. This means running tests again could change the outcome and lead to a different interpretation of the system's performance.

Allowing the model to reason mitigated the issue, but reasoning steps were still different across iterations.

The limited set of 21 crafted test sentences used for evaluation might not be truly representative of the railway domain and the complexity of the German language, which could lead to biased results. Thus, the results can only be seen as an indication for the system's potential.

- **Construct Validity.** The criteria for similarity and prompt to verify test results might not capture all relevant aspects of term similarity, like correctness of domain and context or irrelevant extracted terms.  
Using the strict rate alone may not reflect the actual perception of the results by users.
- **External Validity.** The tests were conducted with a specific model (gpt-4o-mini) and on a specific domain (railway terminology). While large language models generally perform well in different areas, it is not guaranteed that the generated results of the TAS and the validity of the tests can be ensured for other models, model snapshots, languages or domains.

## 8 Conclusion and Future Work

In this chapter, Section 8.1 summarizes the research in this thesis and the extensible Terminology Agent System, capable of extracting terminology and generating definitions from arbitrary texts. Section 8.2 provides potential future work topics and areas that require further research or attention.

### 8.1 Conclusion

This thesis tackled the complex problem of Automatic Term Extraction within the specialized context of Deutsche Bahn, aiming to reduce language barriers faced by foreign workforce. This thesis surveyed the landscape of ATE methodologies - from traditional linguistic and statistical approaches to modern techniques leveraging Large Language Models (LLMs) - and examined the inherent challenges, such as domain dependency, term ambiguity and cross-language compatibility.

The core contribution of this work is the Terminology Agent System (TAS), which employs the blackboard architectural pattern and Generative AI (GenAI). The system's design prioritizes reliability, safety of output and extensibility. The modularity is achieved using an event-driven architecture that invokes different Knowledge Sources, handling specific linguistic tasks like term extraction, normalization and definition generation. This design allows for concurrent execution and

the seamless integration of new algorithms, models and data sources without breaking other parts of TAS.

A product test deployed at Deutsche Bahn showed the system’s efficiency, demonstrating a high recall rate of 91-98% and precision of 78% in term extraction. In 83% of test cases, TAS generated definitions that were evaluated to be safely usable in real-world and practical environments.

The thesis also discussed the challenges of testing applications that employ inherently non-deterministic LLMs, outlining the threats to validity and difficulties when evaluating AI applications.

The thesis put a strong emphasis on the extensibility of the system and reliability of the generated output in German. Thus, cross-language (NFR3) and cross-domain support (NFR4) were not researched in this thesis. Since the tests were performed in an isolated environment, the performance of the system with multiple concurrent users (NFR8) and the actual usability (NFR10) and response time (NFR7) were not evaluated.

## 8.2 Future Work

In this section, potential improvements and future investigations are outlined.

One area for improvement involves integrating the Terminology Agent System (TAS) into existing Deutsche Bahn (DB) systems. This could involve incorporating existing DB Translator KITT (DBTK) into the TAS framework, potentially by adapting DBTK’s linear language pipeline to fit the TAS architecture, thereby creating a unified system. Furthermore, TAS could be extended to enable ontology generation. This would necessitate implementing persistent data resources, such as the KRWD and existing ontologies of the company, as well as developing new Knowledge Sources for generating relations between terms and extending the current data model.

Optimizing the TAS itself presents another possible area for improvement. This could involve fine-tuning pre-trained transformer models like BERT [Dev+19] or T5 [Raf+20], or even larger language models like GPT or Llama. A simple

pipeline could also be designed to enable easy model training for other domains, allowing users to upload documents to generate a more optimized model for their specific field. Prompt optimization is also beneficial: reformulating prompts or exploring different prompting techniques could improve individual Knowledge Sources, while reducing token input and output could lower operational costs. A comprehensive performance analysis is also required, which would involve using more test data, testing different models, and conducting tests with Deutsche Bahn’s official language department. It remains unclear whether TAS can scale effectively with a growing number of users. Future research should investigate its scalability and performance under increased load. The distribution of Knowledge Sources as independent microservices could support increased throughput. Cross-domain and cross-language compatibility were not addressed in this work. Given that LLMs have the potential to handle NLP tasks across multiple languages, it has to be tested and evaluated multilingual scenarios and domains other than railway operations.

The development of a chatbot functionality is another promising direction. This would involve integrating more context, the conversation history, as well as gathering information from external sources relevant to users input. The chatbot should also enable interactive communication, allowing users to ask questions about documents, and receive explanations. Such a feature might require the integration of many different external data sources, especially within complex company structures. Maintaining the correctness of the chatbots output remains an important challenge in this context. Finally, the issue of hallucinations and the non-determinism of LLMs opens another research question. Testing and quality assurance for LLMs are inherently complicated, as traditional static tests have shown to be insufficient in previous experiments: multiple runs of the same test yielded different results. More complex evaluation methods are therefore required. Future research should explore how to make the output of generative AI more deterministic and the potential consequences for AI model performance. Hallucinations, especially those caused by biases in model training data, were not

researched in this thesis but could threaten the correctness of the model’s output [Hua+25]. For instance, when generating a definition, the model might incorporate not only information from the provided text but also pre-trained information that could be incorrect or undesirable due to a redefinition of the term in question.



## List of Figures

Figure 1	The functional model of TAS (UML Use case diagram) .	12
Figure 2	The <b>Extract domain terminology</b> use case. Template from [BD10] .....	13
Figure 3	Analysis object model of TAS (UML class diagram) ....	14
Figure 4	Event taxonomy of all events known by TAS (UML class diagram) .....	15
Figure 5	Taxonomy of all Knowledge Sources responsible for linguistic feature extraction (UML class diagram). ....	16
Figure 6	The flow of events of the blackboard system (UML activity diagram). ....	17
Figure 7	Screenshots of the user interface for the existing DB Translator KITT (left) and the additional terminology information from TAS (right) .....	19
Figure 8	Subsystem decomposition of the Terminology Agent System (UML component diagram). ....	22
Figure 9	The communication between the subsystems and their deployments (UML deployment diagram). <sup>16</sup> .....	25
Figure 10	An example event flow in TAS: <i>White boxes</i> : running KS, <i>Green boxes</i> : starting KS, <i>Transparent boxes</i> : finished KS .....	28
Figure 11	An example for Chain-Of-Thought prompting and a possible result from a model, from [BZ25] .....	32
Figure 12	The prompt used for term extraction. ....	33
Figure 13	The prompt used for term normalization. ....	34

Figure 14	The prompt used for definition generation. ....	35
Figure 15	The prompt used for combining relevant definitions. ....	36
Figure 16	Template code for adding a new Knowledge Source .....	37
Figure 17	Implementation of new Knowledge Source <b>BahnGPTTermExtractor</b> .....	38
Figure 18	Code snippet showing the addition of the newly created Knowledge Sources. ....	39
Figure 19	The implementation of a new Knowledge Source employing the C-Value algorithm. ....	40
Figure 20	Code snippet showing the interface for the KRWD. ....	40
Figure 21	The implementation of the newly introduced <b>KRWD0ccurrenceResolver</b> . ....	41
Figure 22	Add the Knowledge Source in the <b>SessionManager</b> .....	41
Figure 23	Test description for <b>TestTermExtractor_Common</b> .....	43
Figure 24	Test description for <b>TestDefinitionGenerator_Simple</b> ...	43
Figure 25	Layered architecture of TAS showing vertical integration from the user interface to the data layer. Each layer encapsulates a specific role in the system .....	45
Figure 26	Test description for <b>ExtractDomainTerminology_Test</b> . .	45
Figure 27	Excerpt from the system tests. The generated definition has little informational value. ....	49
Figure 28	Excerpt from test results of <b>ExtractDomainTerminology_Test</b> . Every iteration resulted in different probabilities for the result token. ....	50
Figure 29	The prompt used for checking the specificity of a definition for a term. ....	59
Figure 30	Test description for <b>TestDefinitionGenerator_NotEnoughContext</b> .....	59

<b>Figure 31</b>	<b>The prompt used for evaluating the similarity between test result and oracle. ....</b>	<b>59</b>
------------------	---	-----------

# Appendix A: Supplementary Material

User

Ist der folgende Text eine Definition für den Begriff <<term>>? Wenn die Definition spezifisch genug ist, beende deine Folgerung mit TRUE, ansonsten mit FALSE.

<<definition>>

Figure 29: The prompt used for checking the specificity of a definition for a term.

Test case name	TestDefinitionGenerator_NotEnoughContext
Participating objects	OpenAIDefinitionGenerator
Entry conditions	OccurrenceResolved event is passed with the following parameters: <ul style="list-style-type: none"><li>term: “Abstoßen”</li><li>source: “Dies gilt auch für das Abstoßen, sofern in örtlichen Zusätzen nicht Ausnahmen zugelassen sind.”</li></ul>
Expected behaviour	<ul style="list-style-type: none"><li>No definition is generated</li><li>No event is fired</li></ul>

Figure 30: Test description for TestDefinitionGenerator\_NotEnoughContext

Developer

Bewerte die Ähnlichkeit der Ergebnisse der Term Extraktion. Gegeben ist ein Ausgangstext, aus dem Fachbegriffe extrahiert werden mussten.

Der Text ist gegeben. Darunter stehen die erwarteten Begriffe, die extrahiert werden sollten.

Zum Schluss stehen die tatsächlich extrahierten Begriffe.

Bewerte die semantische Ähnlichkeit der extrahierten Begriffe.

Sobald sich ein Begriff grundlegend unterscheidet, beende sofort mit FALSE.

Wenn ein erwarteter Begriff gänzlich fehlt, beende sofort mit FALSE.

Wenn ein Begriff extrahiert wurde, der sicher kein Fachbegriff ist, beende sofort mit FALSE.

Wenn ein Begriff extrahiert wurde, der nicht erwartet wurde, ignoriere diesen. Dies gilt nicht als Unterschied.

Ansonsten Ende sofort mit TRUE.

Bewerte die extrahierten Begriffe.

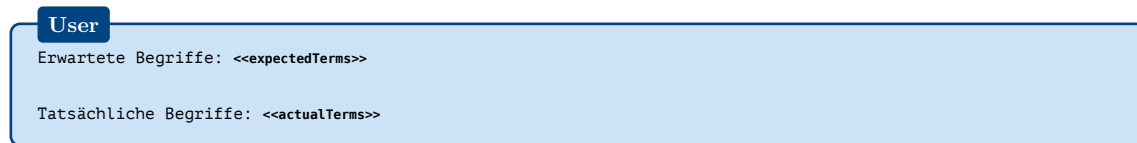


Figure 31: The prompt used for evaluating the similarity between test result and oracle.

---

## Appendix B: Test Results

The results of the performance and system test can be found in the attached file: `figures/test_performance_results.json`

Protocol of the integration test `ExtractDomainTerminology_Test` can be found in the attached file: `figures/protocol_test_integration.txt`

Protocol of the integration test `ExtractDomainTerminology_Test` with reasoning enabled can be found in the attached file: `figures/protocol_test_integration_reasoning.txt`

## Appendix C: Source Code

The implementation of TAS can be found in the attached directory: `tas`

The adaption of TAS for Deutsche Bahn uses confidential resources and thus must be requested at *`cognitive.translation.service@deutschebahn.com`* .

## Bibliography

- [Deu25] Deutsche Bahn AG, "DB Job Portal aktuelle Anzahl an gesuchten Vollzeitstellen." Accessed: Mar. 12, 2025a. [Online]. Available: <https://db.jobs/de-de/Suche>
- [Deu24] Deutsche Bahn AG, "Ersatzverkehr an Der Riedbahn: Busflotte Und Team Sind Komplett." Accessed: Jul. 02, 2025a. [Online]. Available: [https://www.deutschebahn.com/de/presse/pressestart\\_zentrales\\_uebersicht/Ersatzverkehr-an-der-Riedbahn-Busflotte-und-Team-sind-komplett-12879832](https://www.deutschebahn.com/de/presse/pressestart_zentrales_uebersicht/Ersatzverkehr-an-der-Riedbahn-Busflotte-und-Team-sind-komplett-12879832)
- [Til22] Till Uebelacker, "Deutsche Bahn rekrutiert Auszubildende im Ausland," *Süddeutsche Zeitung*, Oct. 2022, Accessed: Jul. 02, 2025. [Online]. Available: <https://www.sueddeutsche.de/wirtschaft/bahn-azubis-migration-1.5672979>
- [dpa24] dpa, "Deutsche Bahn: Fachkräfte im Ausland rekrutieren," *verkehrsrundschau.de*, 2024, Accessed: Jul. 02, 2025. [Online]. Available: <https://www.verkehrsrundschau.de/nachrichten/ausbildungskarriere/deutsche-bahn-fachkraefte-im-ausland-rekrutieren-3517536>
- [Deu25] Deutsche Bahn AG, "Dein Job in Deutschland." Accessed: Mar. 12, 2025b. [Online]. Available: <https://db.jobs/de-de/dein-einstieg/jobs-fuer-nicht-eu-buerger>
- [Eur19] Europäische Kommission, "Durchführungsverordnung (EU) 2019/554 Der Kommission Vom 5. April 2019 Zur Festlegung Praktischer Modalitäten Für Die Erhebung Und Das Format Bestimmter Daten Gemäß Der Richtlinie (EU) 2016/798 Des Europäischen Parlaments Und Des Rates Über Die Eisenbahnsicherheit." Accessed: Jul. 02, 2025. [Online]. Available: <https://eur-lex.europa.eu/legal-content/DE/TXT/PDF/?uri=CELEX:32019R0554>
- [Goe25] Goethe-Institut, "Termine Und Preise - Goethe-Institut Deutschland." Accessed: Jul. 03, 2025. [Online]. Available: <https://www.goethe.de/ins/de/de/tup.cfm?&f=%7B%22languageSubLevel%22%3A%5B%>

5D%2C%22startDate%22%3A%5B%5D%2C%22custom25187177%  
22%3A%5B%5D%2C%22category%22%3A%5B%5D%2C%22eventN  
ameObjectId%22%3A%5B%5D%2C%22filter\_locationId%22%3A%  
5B%22F%2000000501%22%5D%2C%22pspElement%22%3A%5B%5  
D%2C%22age%22%3A%22ER%22%7D

- [DB 25] DB Systel GmbH, “Internal Resource: DB Translator KITT App.” 2025a.
- [Tra+23] H. T. H. Tran, M. Martinc, J. Caporusso, A. Doucet, and S. Pollak, “The Recent Advances in Automatic Term Extraction: A Survey.” Accessed: Mar. 14, 2025. [Online]. Available: <http://arxiv.org/abs/2301.06767>
- [DMS23] G. M. Di Nunzio, S. Marchesin, and G. Silvello, “A Systematic Review of Automatic Term Extraction: What Happened in 2022?,” *Digital Scholarship in the Humanities*, vol. 38, no. Supplement\_1, pp. i41–i47, Jun. 2023, doi: 10.1093/llc/fqad030.
- [Mit22] R. Mitkov, Ed., *The Oxford Handbook of Computational Linguistics*, Second edition. in Oxford Handbooks in Linguistics. Oxford: Oxford University Press, 2022. doi: 10.1093/oxfordhb/9780199573691.001.0001.
- [Bou92] D. Bourigault, “Surface Grammatical Analysis for the Extraction of Terminological Noun Phrases,” in *Proceedings of the 14th Conference on Computational Linguistics* -, Nantes, France: Association for Computational Linguistics, 1992, p. 977. doi: 10.3115/992383.992415.
- [FAM00] K. Frantzi, S. Ananiadou, and H. Mima, “Automatic Recognition of Multi-word Terms: The C-value/ NC-value Method,” presented at the Int. J. on Digital Libraries, Aug. 2000, pp. 115–130. doi: 10.1007/3-540-49653-X\_35.
- [Qin+24] L. Qin *et al.*, “Large Language Models Meet NLP: A Survey.” Accessed: Jul. 22, 2025. [Online]. Available: <http://arxiv.org/abs/2405.12819>

- [BCM24] S. Banerjee, B. R. Chakravarthi, and J. P. McCrae, “Large Language Models for Few-Shot Automatic Term Extraction,” *Natural Language Processing and Information Systems: 29th International Conference on Applications of Natural Language to Information Systems, NLDB 2024, Turin, Italy, June 25–27, 2024, Proceedings, Part I*, vol. 14762. in Lecture Notes in Computer Science, vol. 14762. Springer Nature Switzerland, Cham, pp. 137–150, 2024. doi: 10.1007/978-3-031-70239-6.
- [Gig23] J. Giguere, “Leveraging Large Language Models to Extract Terminology,” in *Proceedings of the First Workshop on NLP Tools and Resources for Translation and Interpreting Applications*, R. L. Gutiérrez, A. Pareja, and R. Mitkov, Eds., Varna, Bulgaria: INCOMA Ltd., Shoumen, Bulgaria, Sep. 2023, pp. 57–60. Accessed: May 31, 2025. [Online]. Available: <https://aclanthology.org/2023.nlp4tia-1.9/>
- [Tra+22] H. T. H. Tran, M. Martinc, A. Pelicon, A. Doucet, and S. Pollak, “Ensembling Transformers for Cross-domain Automatic Term Extraction.” pp. 90–100, 2022. doi: 10.1007/978-3-031-21756-2\_7.
- [Haz+22] A. Hazem, M. Bouhandi, F. Boudin, and B. Daille, “Cross-Lingual and Cross-Domain Transfer Learning for Automatic Term Extraction from Low Resource Data,” in *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, N. Calzolari, F. Béchet, P. Blache, K. Choukri, C. Cieri, T. Declerck, S. Goggi, H. Isahara, B. Maegaard, J. Mariani, H. Mazo, J. Odijk, and S. Piperidis, Eds., Marseille, France: European Language Resources Association, Jun. 2022, pp. 648–662. [Online]. Available: <https://aclanthology.org/2022.lrec-1.68/>
- [BD10] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering: Using UML, Patterns, and Java*, 3rd ed. Boston: Prentice Hall, 2010.
- [Deu25] Deutsche Bahn AG, “DB InfraGO AG.” Accessed: Jul. 09, 2025c. [Online]. Available: <https://www.deutschebahn.com/de/konzern/konzernprofil/Konzernunternehmen/DB-InfraGO-AG-12598696>



- [DB 24] DB InfraGO, “Ril 302.6000 - Grenzüberschreitende Bahnstrecken Mit Frankreich.” Accessed: Feb. 24, 2025a. [Online]. Available: [https://www.dbinfra.go.com/web/schienennetz/netzzugang-und-regulierung/regelwerke/betrieblich-technisch\\_regelwerke/betrieblich\\_technisches\\_regelwerk-12596560#](https://www.dbinfra.go.com/web/schienennetz/netzzugang-und-regulierung/regelwerke/betrieblich-technisch_regelwerke/betrieblich_technisches_regelwerk-12596560#)
- [Gra92] R. B. Grady, *Practical Software Metrics for Project Management and Process Improvement*. Englewood Cliffs, NJ: Prentice Hall, 1992.
- [Jak25] Jakob Nielsen, “Response Time Limits.” Accessed: Mar. 17, 2025. [Online]. Available: <https://www.nngroup.com/articles/response-times-3-important-limits/>
- [DB 25] DB Systel GmbH, “Personal Communication with the Author - Response Time of TAS.” 2025b.
- [Gas+25] J. L. Gastaldi, J. Terilla, L. Malagutti, B. DuSell, T. Vieira, and R. Cotterell, “The Foundations of Tokenization: Statistical and Computational Concerns.” Accessed: Jun. 16, 2025. [Online]. Available: <http://arxiv.org/abs/2407.11606>
- [Thi21] H.-C. Thiel, “Ausgewählte Begriffe und Abkürzungen des Eisenbahn- und Verkehrswesens und der Fahrzeugtechnik sowie ausgewählter Rechtsgrundlagen.” [Online]. Available: [https://bahnsys.uni-wuppertal.de/fileadmin/bauing/bahnsys/2023/Ausgew%C3%A4hlte\\_Begriffe\\_des\\_Eisenbahn-\\_und\\_Verkehrswesens\\_\\_BTU\\_\\_Prof.\\_Thiel\\_\\_24.11.2021.pdf](https://bahnsys.uni-wuppertal.de/fileadmin/bauing/bahnsys/2023/Ausgew%C3%A4hlte_Begriffe_des_Eisenbahn-_und_Verkehrswesens__BTU__Prof._Thiel__24.11.2021.pdf)
- [Lal97] P. Lalanda, “Two Complementary Patterns to Build Multi-Expert Systems,” in *Proceedings of the Pattern Languages of Programs Conference (PLoP'97)*, Monticello, Illinois, USA: Hillside Group / Thomson-CSF Corporate Research Laboratory, Orsay, France, 1997. Accessed: Jul. 20, 2025. [Online]. Available: <https://hillside.net/plop/plop97/Proceedings/lalanda.pdf>

- [DB 25] DB Systel GmbH, “Internal Post: BahnGPT - 'ChatGPT for DB Internal Use'.” [Online]. Available: <https://db-planet.deutschebahn.com/pages/digitalerassistent/apps/content/announcement>
- [Ho+24] A. Ho *et al.*, “Algorithmic Progress in Language Models.” Accessed: Jul. 24, 2025. [Online]. Available: <http://arxiv.org/abs/2403.05812>
- [Goo25] Google AI for Developers, “Structured Output.” Accessed: Jul. 13, 2025. [Online]. Available: <https://ai.google.dev/gemini-api/docs/structured-output>
- [Jet24] JetBrains, “Kotlin Programming Language.” [Online]. Available: <https://kotlinlang.org/>
- [Goo24] Google, “Jetpack Compose.” [Online]. Available: <https://developer.android.com/jetpack/compose>
- [Hug24] Hugging Face, “Transformers: State-of-the-art Natural Language Processing for PyTorch and TensorFlow.” [Online]. Available: <https://huggingface.co/docs/transformers/index>
- [Exp24] Explosion, “spaCy: Industrial-strength Natural Language Processing in Python.” [Online]. Available: <https://spacy.io/>
- [Doc24] Docker Inc., “Docker: Enterprise Container Platform.” [Online]. Available: <https://www.docker.com/>
- [The24] The Kubernetes Authors, “Kubernetes: Production-grade Container Orchestration.” [Online]. Available: <https://kubernetes.io/>
- [Dee24] G. DeepMind, “Gemini Language Model.” [Online]. Available: <https://deepmind.google/technologies/gemini/>
- [Met24] Meta AI, “LLaMA: Large Language Model Meta AI.” [Online]. Available: <https://ai.meta.com/llama/>
- [Ant24] Anthropic, “Claude Language Model.” [Online]. Available: <https://www.anthropic.com/index/claude>
- [Git24] GitLab Inc., “GitLab: DevOps Platform.” [Online]. Available: <https://about.gitlab.com/>

- [DB 25] DB Systel GmbH, “Internal Document: Docker Base Images for DB.”
- [BZ25] J. Berryman and A. Ziegler, *Prompt Engineering for LLMs: The Art and Science of Building Large Language Model-Based Applications*, First Edition. Sebastapol, CA: O'Reilly Media, 2025.
- [Sch+25] S. Schulhoff *et al.*, “The Prompt Report: A Systematic Survey of Prompt Engineering Techniques.” Accessed: Jul. 02, 2025. [Online]. Available: <http://arxiv.org/abs/2406.06608>
- [Wei+23] J. Wei *et al.*, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.” Accessed: Jul. 02, 2025. [Online]. Available: <http://arxiv.org/abs/2201.11903>
- [Ant25] Anthropic, “Chain Complex Prompts for Stronger Performance.” Accessed: Jul. 04, 2025. [Online]. Available: <https://docs.anthropic.com/en/docs/build-with-claude/prompt-engineering/chain-prompts>
- [Ope25] OpenAI, “Strategy: Split Complex Tasks into Simpler Subtasks.” Accessed: Jul. 20, 2025. [Online]. Available: <https://platform.openai.com/docs/guides/prompt-engineering/tactic-specify-the-steps-required-to-complete-a-task#strategy-split-complex-tasks-into-simpler-subtasks>
- [Ren24] M. Renze, “The Effect of Sampling Temperature on Problem Solving in Large Language Models,” in *Findings of the Association for Computational Linguistics: EMNLP 2024*, Miami, Florida, USA: Association for Computational Linguistics, 2024, pp. 7346–7356. doi: 10.18653/v1/2024.findings-emnlp.432.
- [Pat+24] D. Patel *et al.*, “Exploring Temperature Effects on Large Language Models Across Various Clinical Tasks.” Accessed: Jun. 29, 2025. [Online]. Available: <http://medrxiv.org/lookup/doi/10.1101/2024.07.22.24310824>
- [DAI24] DAIR.AI, “LLM Settings | Prompt Engineering Guide.” Accessed: Jul. 13, 2025. [Online]. Available: <https://www.promptingguide.ai/introduction/settings>

- [Hol+20] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, “The Curious Case of Neural Text Degeneration.” Accessed: Jul. 13, 2025. [Online]. Available: <http://arxiv.org/abs/1904.09751>
- [Mat+24] Mathieu Fenniak *et al.*, “The pypdf Library.” [Online]. Available: <https://pypi.org/project/pypdf/>
- [Aue+25] C. Auer *et al.*, “Docling: Ein Effizientes Open-Source-Toolkit Für KI-gesteuerte Dokumentenkonvertierung.” [Online]. Available: <https://arxiv.org/abs/2501.17887>
- [Deu24] Deutsche Bahn AG, “Internal Post: Die Weichen Für Generative KI in Der DB Wurden Gestellt!.” [Online]. Available: <https://db-planet.deutschebahn.com/pages/11e7697d-2ac7-4b9a-89e2-6ffef9566ac6/apps/blog/4fb66e9a-be15-4c9f-9fbd-c61f679dbdf9/view/b2024f04-eee5-42ad-bd41-fbb082627240>
- [Lu21] K. Lu, “Kevinlu1248/Pyate: Python Automated Term Extraction.” Jun. 2021. doi: 10.5281/zenodo.5039289.
- [Deu25] Deutsche Bahn AG, “Internal Resource: Konzernregelwerksdatenbank.” 2025d.
- [Ati+25] B. Atil *et al.*, “Non-Determinism of "Deterministic" LLM Settings.” Accessed: Jul. 13, 2025. [Online]. Available: <http://arxiv.org/abs/2408.04667>
- [DB 24] DB InfraGO AG, “Fahrdienstvorschrift: Richtlinien 408.21–27 Und 408.48 (Handbuch 40820),” Frankfurt am Main, Dec. 2024b. [Online]. Available: <https://www.dbinfrago.com/.../Handbuch-40820-data.pdf>
- [Wik] Wikipedia contributors, “DB-Baureihe 420 – Wikipedia.” [Online]. Available: [https://de.wikipedia.org/wiki/DB-Baureihe\\_420](https://de.wikipedia.org/wiki/DB-Baureihe_420)
- [Lev66] V. I. Levenshtein, “Binary Codes Capable of Correcting Deletions, Insertions and Reversals,” *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

- [Mon+25] S. Mondal *et al.*, “Exploring Methodologies for Computing Sentence Similarity in Natural Language Processing,” in *Advances in Data and Information Sciences*, S. Tiwari, M. C. Trivedi, M. L. Kolhe, and B. K. Singh, Eds., Singapore: Springer Nature, 2025, pp. 251–261. doi: 10.1007/978-981-97-9619-9\_21.
- [Hus+23] M. Hussain, U. U. Rehman, T. D. Nguyen, and S. Lee, “CoT-STS: A Zero Shot Chain-of-Thought Prompting for Semantic Textual Similarity,” in *2023 6th Artificial Intelligence and Cloud Computing Conference (AICCC)*, Kyoto Japan: ACM, Dec. 2023, pp. 135–139. doi: 10.1145/3639592.3639611.
- [Hua+25] L. Huang *et al.*, “A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions,” *ACM Transactions on Information Systems*, vol. 43, no. 2, pp. 1–55, Mar. 2025, doi: 10.1145/3703155.
- [Dev+19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Association for Computational Linguistics, 2019, pp. 4171–4186. [Online]. Available: <https://aclanthology.org/N19-1423/>
- [Raf+20] C. Raffel *et al.*, “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer.” [Online]. Available: <https://arxiv.org/abs/1910.10683>