Heidelberg University
Institut of Computer Science
Data Science Group

Master's Thesis

Knowledge Acquisition and Question Answering with Expert Systems using LLMs

Name:	Jonas Gann
Matriculation Number:	3367576
Supervisor:	Prof. Dr. Michael Gertz
Submission Date:	August 5, 2025

Hiermit versichere ich, dass ich die Arbeit selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und wörtlich oder inhaltlich aus fremden Werken Übernommenes als fremd kenntlich gemacht habe. Ferner versichere ich, dass die übermittelte elektronische Version in Inhalt und Wortlaut mit der gedruckten Version meiner Arbeit vollständig übereinstimmt. Ich bin einverstanden, dass diese elektronische Fassung universitätsintern anhand einer Plagiatssoftware auf Plagiate überprüft wird.

Abgabedatum: August 5, 2025

Zusammenfassung

Abstract

TODO: Get inspiration from problem description and visualization of Jr, Li, and Tee 2025

Contents

1 Introduction

Medical information such as diagnosis or therapy of diseases is usually available as a running text. This unstructured information is suitable for human comprehension but is difficult to process with a computer. Approaches in Artificial Intelligence such as Large Language Models, while achieving impressive results in medical question answering tasks ...

2 Background and Related Work

2.1 Expert Systems

2.1.1 MYCIN

Figure 2.1 shows the architecture of the system of a expert system for medical diagnosis called "MYCIN" (Buchanan and Shortliffe 1984) developed in the 1970s.

2.2 Prolog

Definition 2.1. *(Prolog Knowledge Base). A Prolog Knowledge Base is defined as:*

$$kb := \langle f_1, f_2, ..., f_n, r_1, r_2, ..., r_k \rangle$$

where f_n denotes an individual Prolog fact and r_k denotes an individual Prolog rule.

The Prolog Knowledge Base is expected to consist of valid Prolog rules and facts.

Definition 2.2. *(Prolog Rule). A Prolog Rule is defined as:*

$$r := h \ :\text{-}\ b_1, b_2, ..., b_m$$

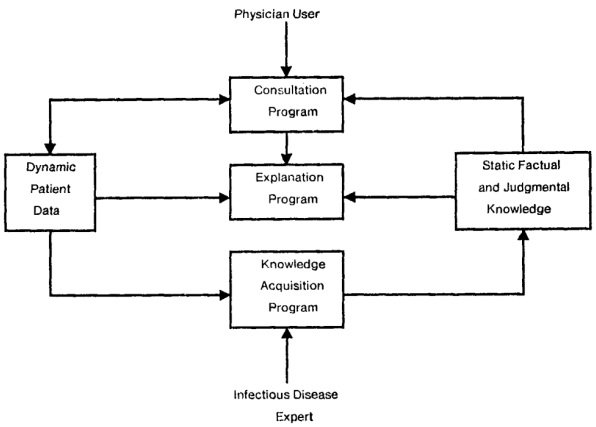

Figure 2.1: MYCIN Architecture Overview

where h is the head of the rule (a single Prolog goal), and $b_1, b_2, ..., b_m$ is the body consisting of one or more goals that must be satisfied for h to be true.

Definition 2.3. *(Prolog Fact). A Prolog Fact is defined as:*

$$f := p(t_1, t_2, ..., t_n)$$

where p is a predicate symbol and each t_i is a term (e.g., a constant or a variable). A fact declares that the predicate p holds for the given terms.

2.3 Retrieval Augmented Generation

2.4 Medical Data Sources

2.4.1 Medical Guidelines

2.4.2 ICD10

2.4.3 MeSH

2.5 Related Work

3 The Prolog Expert System

This chapter presents the methods and techniques used for the development of a system for automated knowledge acquisition, focusing on the formalization of domain knowledge using the Prolog language. Possible approaches for the utilization of Prolog knowledge bases in the context of expert systems for question answering are presented.

3.1 Overview and Objectives

An important drawback of modern AI systems, especially LLMs, is that they largely operate as a black box. It is not fully understood how information is stored within neural networks of, for example, the transformer architecture or how such a model accesses and processes information to reason about questions (Chen et al. 2025). During training, models learn approaches to solving tasks in ways that are not understandable or verifiable by a human. This leads to a variety of issues.

Without being able to understand the processes of how an LLM accesses and processes learned information, it is challenging to determine whether an answer is factually correct. There are a variety of approaches to mitigate this problem. One of the most prominent examples are retrieval-based systems which focus on adding relevant domain knowledge during inference time. This is usually done by adding the retrieved text to the user prompt. An important advantage of this approach is that the information retrieved during the generation of the answers can be used to manually verify the accuracy of the answer. However, following the retrieval-based approach above, some key challenges remain:

Correctness: Although retrieval-based approaches help users verify the correctness of the information, it is usually not feasible to guarantee that the responses generated by the system are correct. It is possible that the domain knowledge retrieved by the system is not relevant to the user question, that the retrieved information does not contain all relevant context, or that the LLM does not faithfully reflect the information within the retrieved data.

Reasoning: Retrieval-based approaches often rely on inference-time reasoning using chain-of-thought approaches to solve complex user questions. This approach empirically

improves the accuracy of the answers; however, research suggests that the reasoning steps generated as part of the answer do not faithfully reflect the actual reasoning processes performed by the LLM (Chen et al. 2025). Hence, it remains a challenge to faithfully determine and explain the reasons why an LLM generated an answer. As inference-time reasoning using chain-of-thought prompts works within the black-box of an LLM, no improvements regarding correctness guarantees are made.

Determinism: An important challenge of the probabilistic nature of LLMs is that their behavior in real-life scenarios is difficult to predict or evaluate. Especially in high-risk applications such as medical scenarios, it is important to guarantee appropriate behavior in all possible scenarios.

Transparency: Another challenge related but not identical to correctness is transparency. The lack of transparency in many machine learning approaches exacerbates the issues of correctness. If a model behaves incorrectly in a transparent manner, e.g. the processing steps which lead to the answer are understandable by a human, identifying a problem becomes much easier. This is an essential property to mitigate problems regarding correctness in a real-life application, as it enables users to better decide whether to trust an answer or not. As previously mentioned, RAG systems already increase the level of transparency by referencing source material used for answer generation. Nevertheless, even if a document was referenced by the model, it is not guaranteed, that the answer of the model matches the content of the document. Hence, an approach which makes the answer generation process itself transparent would be beneficial.

Maintainability: Maintainability means the ease and granularity of updating the behavior of the model. Expanding upon transparency, maintainability is an important property to correct errors, add new information, or update outdated information. This enables for a process of incremental improvements, continuously adapting a system to real-life problems. Modern machine learning approaches are not easy to maintain. Often, large quantities of data are necessary for a model to learn new information through a resource intensive training or finetuning process. Additionally, the influence new training data has on the model as a whole in respect to side-effects, is difficult to predict. An alternative approach enabling the modification of individual pieces of information with predictable effects on the behavior of the system would be beneficial. RAG-systems improve maintainability by enabling models to access a repository of documents that are relevant for a given question. The information reflected by the model can then be updated by modifying information within the underlying documents. An important issue of this approach is the complexity of retaining a high quality data source within a large and growing corpus of documents. Another important issue is, that developing a functioning retrieval pipeline for a RAG-system that accurately retrieves relevant

information with necessary context is a challenging and error prone task. It would be advantageous to be able to directly update unique pieces of information that determine the behavior of the system.

In order to tackle the challenges described above, an alternative system with design-time retrieval and reasoning is presented. In contrast to existing retrieval-based systems where context is retrieved and reasoning is performed dynamically for a given question, a system is proposed which focuses on precomputing an information representation that contains all relevant retrievable information as well as all relevant reasoning processes. This can be viewed as a general purpose retrieval task aimed at capturing relevant information and reasoning approaches for future questions. The goal is to represent the retrieved information as Prolog rules and facts. This logic-based representation of information natively incorporates the capability to express complex reasoning processes found within the source text. Prolog runtimes enable the utilization of **transparent** and **maintainable** Prolog knowledge bases with advanced querying capabilities, providing **explainable**, **deterministic**, and **correct** query results. Applying this methodology on a knowledge base that consists of a general purpose retrieval has the capability of solving the previously mentioned objectives.

This approach of formalizing knowledge into a technical language such as Prolog is not new, and the advantages regarding correctness, reasoning, determinism, transparency, and maintainability were a motivating factor for their development in the past. This category of technical approaches can be called "rule-based expert systems". The main challenge in construction of an expert system was the fact that knowledge bases had to mostly be handcrafted by domain- and technical experts. This endeavor turned out to be very time consuming as the communication between domain- and technical experts tended to be inefficient and the technical modeling of information was complex. In order to leverage existing know-how within the research domain of expert systems, the thesis will look at approaches for automating the challenging knowledge acquisition task of expert systems while taking inspiration from the design of existing expert system architectures.

The "MYCIN" (Buchanan and Shortliffe 1984) system presented in Section 2.1.1 can be used as a reference architecture for the design of an expert system. As part of this thesis, this reference architecture is modified, and the individual components are implemented utilizing machine learning technologies and approaches suitable for the fully automated construction of a knowledge base and utilization as a natural language question answering system.

Figure 3.1, visualizes the newly proposed architecture. It distinguishes between a "Knowledge Acquisition Pipeline" operated by a system designer and a "Question An-

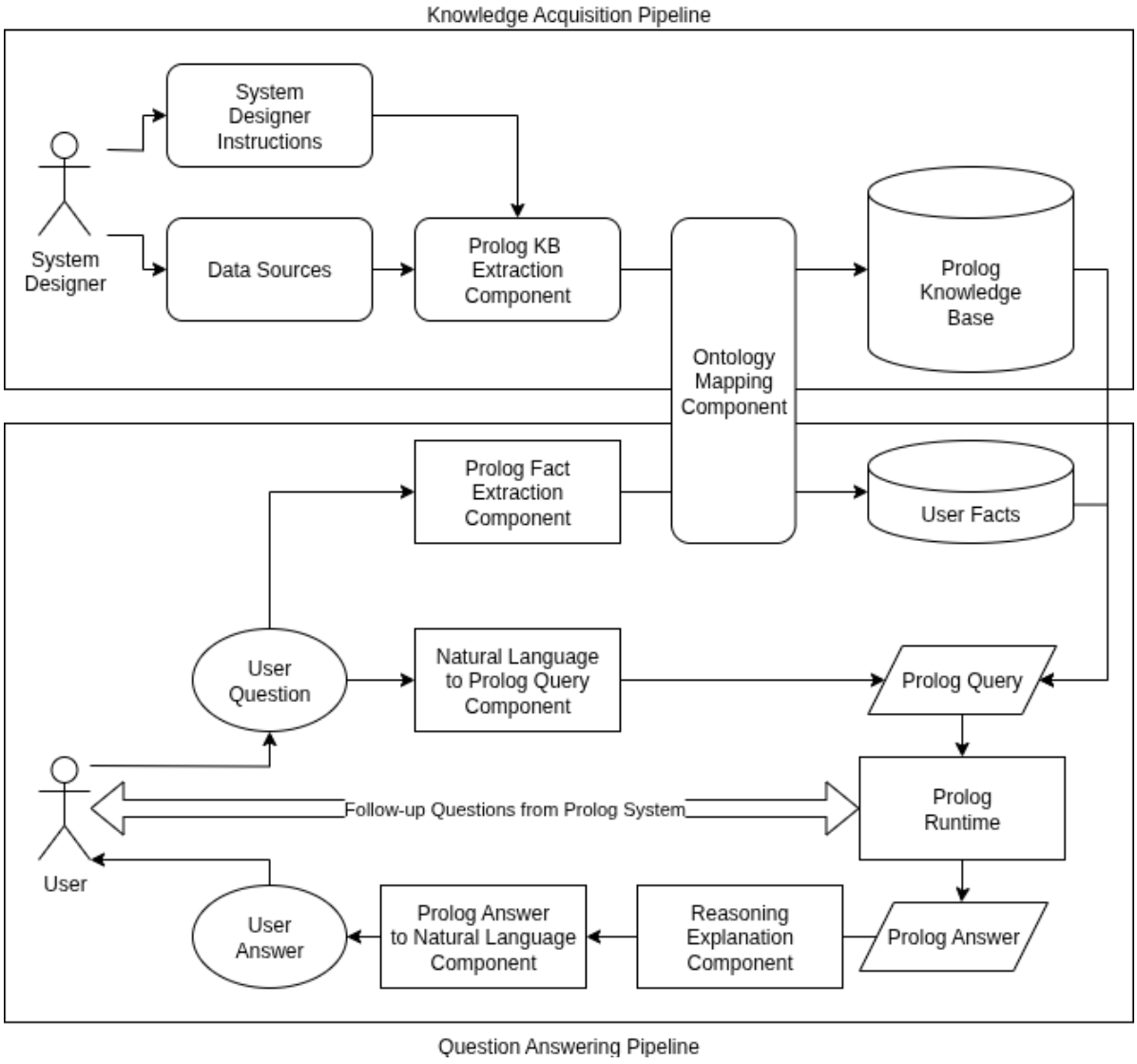

Figure 3.1: System Overview

swering Pipeline" operated by the user. The goal of the System Design Pipeline is the construction of a Prolog Knowledge Base for a given use case. The use case is specified through the system designer by selecting relevant text data, as well as specifying requirements of the Prolog system. This involves the specification of user questions that the system needs to be able to answer, as well as the format of follow-up questions that the system can ask the user.

The central component of the system design pipeline is the "Prolog Knowledge Base Extraction Component". The task of this component is to construct a Prolog Knowledge Base given the instructions by the system designer. This thesis leverages Large Language Models as well as various prompting techniques in order to achieve a reliable automated process for generating a Prolog Knowledge Base. This component closely resembles the "Knowledge Acquisition Program" of the reference architecture. An important goal of the thesis is to automate this module as much as possible using modern machine learning techniques. Another relevant distinction to the reference architecture is the fact that the Knowledge Acquisition Pipeline does not rely on domain experts, but aims to automate the knowledge acquisition to an extent that technical experts with limited domain knowledge (called system designer) are capable of configuring the system appropriately.

The Prolog knowledge base, which is constructed as a result of the Knowledge Acquisition Pipeline, resembles the "Static Factual and Judgmental Knowledge" component of the reference architecture. The main difference from the reference architecture is the explicit utilization of Prolog as a formalization language for knowledge.

The question answering pipeline consists of multiple smaller components in order to make the Prolog Knowledge Base utilizable by a user as a question answering system. The "Natural Language to Prolog Query Component" translates the question of the user into a syntactically correct Prolog query which is suitable for execution on the existing Prolog Knowledge Base. A Prolog Runtime is used to execute the Prolog query on the given Prolog Knowledge Base. A "Prolog Answer to Natural Language Component" translates the Output of the Prolog Runtime to an answer which matches the originally stated user question. In order to answer questions specific to the situation of the user, two approaches for leveraging user information are available. First, relevant information provided within the question of the user can be extracted as Prolog facts using the "Prolog Fact Extraction Component". In case the Prolog system requires additional information in order to determine an answer, follow-up questions can be dynamically prompted by the Prolog system and answered by the user. These components together closely resemble the "Consultation Program" of the reference architecture which aims to be the interaction point of the user to the expert system, on one side dynamically

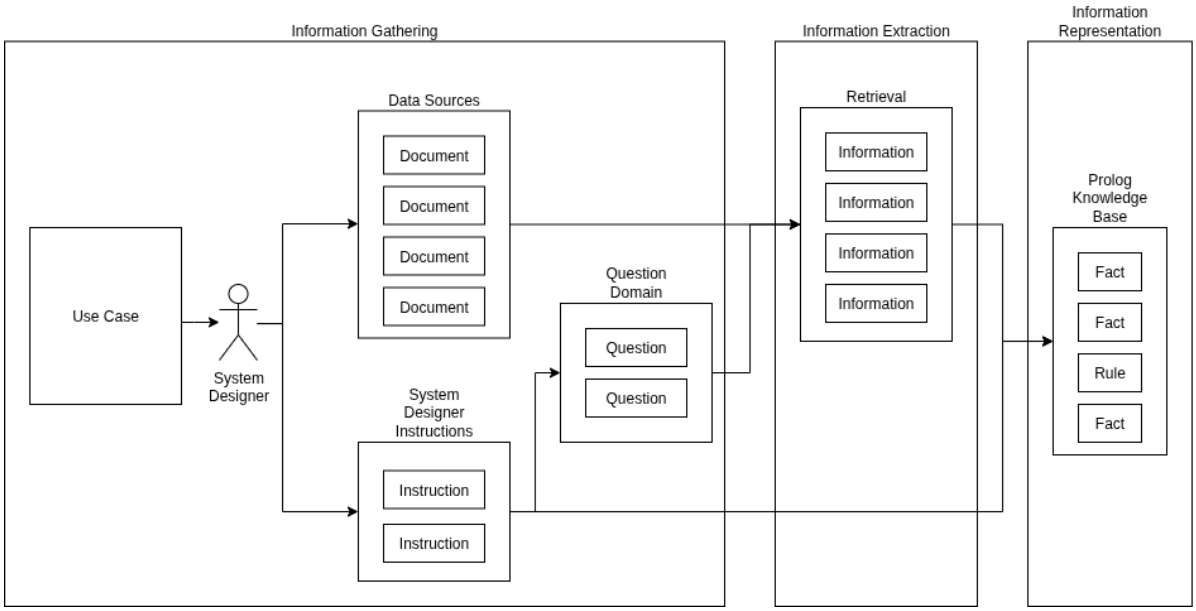

Figure 3.2: Data Processing Pipeline

storing patient information on the other side evaluating questions by referencing the formalized knowledge representation.

A "Reasoning Explanation Component" summarizes Prolog reasoning steps as an answer explanation. This enables the user to inspect all relevant reasons which lead the system to a given answer. This component resembles the "Explanation Program" of the reference architecture.

To bridge the terminology gap between the Prolog system and the user, a "Ontology Mapping Component" acts as a translation layer between textual medical information and its representation within the Prolog Knowledge Base. Its main purpose is to map medical terms to standardized medical ontologies on a sufficiently abstract level. This is necessary for the Prolog System and the user to communicate using agreed upon terminology. This component is not part of the reference architecture.

3.2 The Knowledge Acquisition Pipeline

This section formulates the task the Knowledge Acquisition Pipeline aims to solve on a conceptual level. Knowledge acquisition can be defined as "[t]he transfer and transformation of problem-solving expertise from some knowledge source to a program [...]" (Buchanan and Shortliffe 1984, p. 150). Figure 3.2 visualizes the previously introduced knowledge acquisition pipeline in more detail. The pipeline can be separated into three phases, the information gathering phase, the information extraction phase, and

the information representation phase.

In the information gathering phase, the goal is for a system designer to provide the information required to specify the content and purpose of the system. The content of the system is sourced from a corpus of documents provided by the system designer.

Definition 3.1. *(Document). A document is defined as a tuple:*

$$d := \langle e_1, e_2, ..., e_n \rangle$$

where e_i denotes an individual element within the document. This element can be either a sentence (text string) or an image (e.g., JPG).

The use case (meaning the purpose of the system) is specified by the system designer's instructions. These instructions can be constructed in a variety of ways and are explained in more detail in the following sections. Their purpose is to guide the information extraction and information representation phase to match the use case of the system. To guide the information extraction phase, the instructions determine the creation of user question samples that are intended to closely resemble the targeted usage of the system. User questions can consist of multiple separable subquestions as well as facts that describe context information about the user.

Definition 3.2. *(User Question). A question posed by a user.*

$$uq := \langle f_1, f_2, ..., f_n, q_1, q_2, ..., q_k \rangle$$

where f_i denotes factual information about the user in the form of a string and q_j denotes an individual question which can be answered separately from all other questions.

The set of user questions defines the question domain by serving as a representative sample drawn from the potentially infinite space of questions generated by the use case. To guide the information representation phase, instructions from the system designer can be directly incorporated as text in the LLM prompts.

Definition 3.3. *(Question Domain). Potentially infinite space of user questions regarding a use case:*

$$Q := \langle q_1, q_2, ..., q_n \rangle$$

where q_i denotes an individual question.

The information extraction phase can be viewed as a general-purpose retrieval task. Each document contains a multitude of information, of which only a subset is relevant for

a targeted system usage, which is specified by the question domain. Hence, information that is relevant for answering questions from the question domain is retrieved, in effect filtering the information from the document corpus for the specific use case at hand.

Definition 3.4. *(Retrieval). The selection of document information relevant for a user question.*

$$ret(d, uq) := \langle i_1, i_2, ...i_k \rangle$$

where d denotes a document, uq denotes a user question, and i_k denotes a piece of document information.

The information representation phase can be viewed as general-purpose question answer generation. The goal is to formulate answers to not yet specified questions within a structure that is flexible enough to be applied to a multitude of question scenarios. In Prolog this can be done by defining facts and rules in such a way that the underlying logic of a given use case is captured. It is the separation of domain-specific knowledge from the usage of the knowledge that has been found to give great flexibility (Buchanan and Shortliffe 1984, p. 149) in answering a variety of unknown questions through the means of reasoning. Knowledge Base Extraction describes the process of representing retrieved information in the form of a Prolog Knowledge Base according. The retrieval can contain facts such as typical symptoms of diseases, but also contains rules like for example thresholds for laboratory values for medical diagnosis. The goal is to encode both facts and rules of the retrieval within a Prolog Knowledge Base to formalize available reasoning processes that can be used to answer questions.

Definition 3.5. *(Knowledge Base Extraction). Given a retrieval and system designer instructions, a Knowledge Base Extraction refers to the construction of a Prolog Knowledge Base and is defined as:*

$$kbe(ret(uq), sdi) = kb \quad \forall uq \in Q$$

where $ret(uq)$ denotes the relevant information retrieved from a document based on a user query $uq \in Q$, sdi denotes system designer instructions and kb denotes the resulting Prolog Knowledge Base.

3.2.1 Information Gathering

This section presents challenges and solution approaches during the information gathering phase.

Challenge 1.1: Definition of the Use Case
Determine the use case of the system.

Expert systems are an approach to structuring knowledge by defining facts and rules in a declarative manner, resulting in emergent reasoning capabilities that simulate expert decision-making. Not all real-life problems and scenarios are naturally suitable for being structured in this way. Randall Davis and Jonathan J. King (Buchanan and Shortliffe 1984, p. 28-30) describe in three dimensions the use cases that are suitable for representation as an expert system.

1. Use cases that consists of many states (vs. use cases that are best being described as concise unified theory)
2. Use case that have a control flow that consists of independent actions (vs. use cases that have a control flow that consists of multiple parallel interdependent processes)
3. Use cases that have a declarative representation (vs. use cases that have a procedural representation)

In addition, the utilization of Prolog as an engine to solve reasoning tasks limits the number of questions the system can reasonably handle. Prolog queries can be specified in two ways: Either a statement is evaluated regarding its truth value (true or false), or a concrete set of values has to be determined for which a query evaluates to true. As user questions are translated into Prolog queries, a translation can only then be accurate if the user question adheres to the two question modes described above. Another limitation of the way Prolog evaluates queries comes from the closed-world assumption, which states that all information that cannot be evaluated to true is defined as false. This means that questions that are out of scope of the system and its domain of knowledge must not be asked, or better yet, rejected.

This thesis will cover the use case of **medical diagnosis**. The goal of the system is to be able to answer questions from non-medical users about their health status, especially with respect to the diagnosis of possible medical conditions based on provided context information such as symptoms and laboratory results. Considering the aforementioned criteria by Randall Davis and Jonathan J. King, this use case is suitable for representation in an expert system because the use case of medical diagnosis contains a multitude of states comparable to a patient file, a straightforward control flow that evaluates conditions necessary for diagnosis, and a declarative representation of said

conditions. In addition, the use case of diagnosis matches the querying methodology of Prolog in evaluating if a claimed medical condition is plausible or in evaluating plausible medical conditions.

Challenge 1.2: Definition of the Question Domain
Describe the expected information that the system should contain and the questions it should be able to answer.

Once a use case has been determined, it should be specified what information is within the scope of the system. Some representation has to be found which describes the targeted set of user questions that the system should be capable of answering. This is important to distinguish between information from the data source that should be formalized within the system and information that can be left out.

Question category description

There are multiple approaches to describe the question domain. The most straightforward approach would be a textual description of the types of questions that can be expected. This textual description can be added to the context of LLMs during the Prolog knowledge base creation process to tailor the structure and content of the knowledge base to the given specifications. To efficiently describe a question domain in flat text, one can make use of existing approaches to categorization of question answering datasets and the description of the datasets therein. For example, Rogers, Gardner, and Augenstein (2021) provide a study on question answering datasets and a variety of formats along which question, answer, and evidence types can be differentiated. Inspired by their work, the following properties of question categories can be defined with a Prolog question answering system in mind:

```
Question Category:              Title of the category

Expected answer:                Expected answers within the category

Prolog query format:            Either "Instantiation of variables" or
                                "evaluation of truth value"

Possible context information:   Information that a user should provide
                                as part of the question
```

`Question patterns:`	`Cloze texts describing the general structure of the question category`
`Examples:`	`Question examples specific to the domain knowledge`

An example of a question category can look as follows:

```
Question Category:
Diagnostic Tests

Expected answer:
Test names or procedures (e.g., fasting glucose test, OGTT)

Prolog query format:
Instantiation of variables

Possible context information:
Disease or condition

Generative templates:
What tests are used to diagnose ⟨condition⟩?
Which procedures are relevant for confirming ⟨disease⟩?

Examples:
What test is used to confirm Type 2 Diabetes?
How is diabetes diagnosed?
```

Synthetic QA Category creation

Manually defining question categories is an ideal approach when the use case of the system is very specific and the number of relevant question categories is small. Otherwise, approaches for automatically generating question categories are advantageous. To this end, LLMs can be tasked with creating question categories according to the previously specified template based on provided context information. For the case of medical diagnosis, a possible prompt can look as follows:

Prompt 3.1. *Your task is to create <number categories> question categories according to the following template and domain knowledge. Based on the categories, synthetic*

questions will be created. The question categories must not be specific to a disease. <question category template> <domain knowledge>

Synthetic creation of QA datasets

Each question category describes a possibly infinite number of questions. To better operationalize these categories, e.g. for the purpose of information retrieval, synthetic questions can be generated. Using the descriptions of the categories, an LLM can be prompted to generate a diverse set of possible user questions. One can add domain knowledge to the context of the LLM and instruct that the synthetic questions have to be answerable by the domain knowledge. The effect of these two approaches will be further discussed in the context of information filtering and evaluation.

A possible prompt can look as follows:

Prompt 3.2. *Your task is to create <number of questions> questions that match the following question template: <template>. Your questions have to be answerable with the following content: <domain knowledge>*

Existing QA datasets

Instead of synthetically generating a question dataset, one can search for existing question datasets online that match the use case. Alternatively, question categories can be used to filter an existing question dataset by attributing matching questions to a question category.

Challenge 1.3: Determine the data sources
Determine the important criteria of the data sources that are used to extract the necessary domain knowledge. Find suitable data that match the criteria as well as possible. Examples of generally applicable criteria are:

- The data is in a format that is suitable for the expected digital processing steps
- The data originates from a trusted source
- The information within the data is sufficient for answering the questions of the question domain
- The information within the data is on a complexity level that is suitable for the use case
- The text is formulated in a language and is using terminology that can be understood by the targeted user base

Once the use case and question domain are clear, data sources have to be found which can be used as the source of information for the system. Depending on the use case different key characteristics can be necessary. An important characteristic is the ability to digitally process a data source or the availability of necessary of preprocessing solutions. This thesis focuses on data in the form of text documents with optional visual information within the text in form of diagrams or tables. There exist many data formats for documents such as pdf, txt, docx, md and more. Some data formats may require specialized tools such as optical character recognition to represent the document as raw text. It is also important, that the data originates from a trusted source. As information contained within the selected documents will be used as the ground truth within the system and answers will be generated based on the facts and rules extracted from the document, any false or misleading data within the source will be reflected in the answers of the system. Given a use case and a question domain, the selected documents should contain all necessary information that is required to answer expected questions at a sufficiently complex level. Later processing steps aim to filter information that is relevant for the defined question domain. However, given a relevant concept, the system will aim to formalize all information concerning this concept with the same level of detail as it occurs in the document. Hence, a difference between the complexity of available and targeted information would only be permissible if the source text is more detailed than required. In this case, approaches can be employed that summarize and

abstract the information to be suitable for the targeted user base (see Challenge 4.2). Similarly, it is of importance to determine how the information is formulated within the text document. The terminology used in the document has to be understandable to the target audience. Otherwise, a translation or simplification may be required (see Challenge 4.2).

For the use case of medical diagnosis, a variety of data sources are available. Some interesting ones are shown in Table 3.1.

TODO: finish table (more meta information about data source)

Data Source	**Notes**
`awmf.org`	Detailed brochures (ca. 200 pages) about many medical topics
`gesund.bund.de`	Diseases A–Z with ICD codes, detailed structured articles
`gesundheitsinformation.de`	Articles about medical topics
`apotheken-umschau.de`	Articles about ca. 400 diseases
`bundesaerztekammer.de`	Short brochures (ca. 2 pages)
`stiftung-gesundheitswesen.de`	Articles about diseases A–Z

Table 3.1: Medical Diagnosis Data Sources

The thesis focuses on the data source "awmf.org", which contains officially recognized medical guidelines for medical practitioners in Germany. This gives the data a very high level of trust, which is necessary for a system that aims to answer questions within the medical domain. Structured by medical subjects, medical guidelines exist for a large amount of medical topics such as diseases, medical procedures, preventative measurements, and many more.

This thesis focuses on the use case of medical diagnosis, hence medical guidelines for diseases which contain chapters about their diagnosis are of importance. The documents can be downloaded as pdf files with embedded text data. However, preprocessing is necessary to extract text data from the document in a correctly structured manner (see Challenge 2.2). The document is written in German and is aimed at medical professionals. Therefore, the text uses terminology that cannot always be expected to be understood by laymen. Alternative data sources such as "apotheken-umschau.de" that focus on the explanation of medical topics in laymen terms would be more suitable in this regard; however, as a result of this, they lack the detail and precise description of concepts that are advantageous for being used as the basis of reasoning steps for the purpose of medical diagnosis. (TODO: More details and examples). An ideal solution in this context is the utilization of detailed medical guidelines for internal reasoning about possible diagnosis and the communication of results, as well as a summary of the

reasoning process, to the user in understandable language.

Challenge 1.4: Data Preparation
TODO: make concrete
What are necessary preprocessing steps for the data sources?

TODO: describe document
TODO: describe necessary preprocessing steps

3.2.2 Information Selection

This section presents challenges and solution approaches during the information selection phase.

Challenge 2.1: Filtering of relevant information
Identify and filter information relevant for the question domain.

The data sources of the domain knowledge can be very extensive. However, the question domain can be limited to only a small portion of information distributed among documents. In order to reduce the amount of noise resulting from information unrelated to the question domain, relevant parts of the data sources can be extracted for the purpose of creating a knowledge base.

Use case filtering

The description of the use case can be used to generate a summary of the source documents focusing only on relevant information. This summary can be generated by an LLM. A possible prompt can look as follows:

Prompt 3.3. *Your task is to generate a summary of the following document text, focusing only on information that is relevant for the following use case. Use case: <use case>. Document: <document>*

Question category filtering

- Retrieval based on question category description
- Iterating text and marking relevant passages

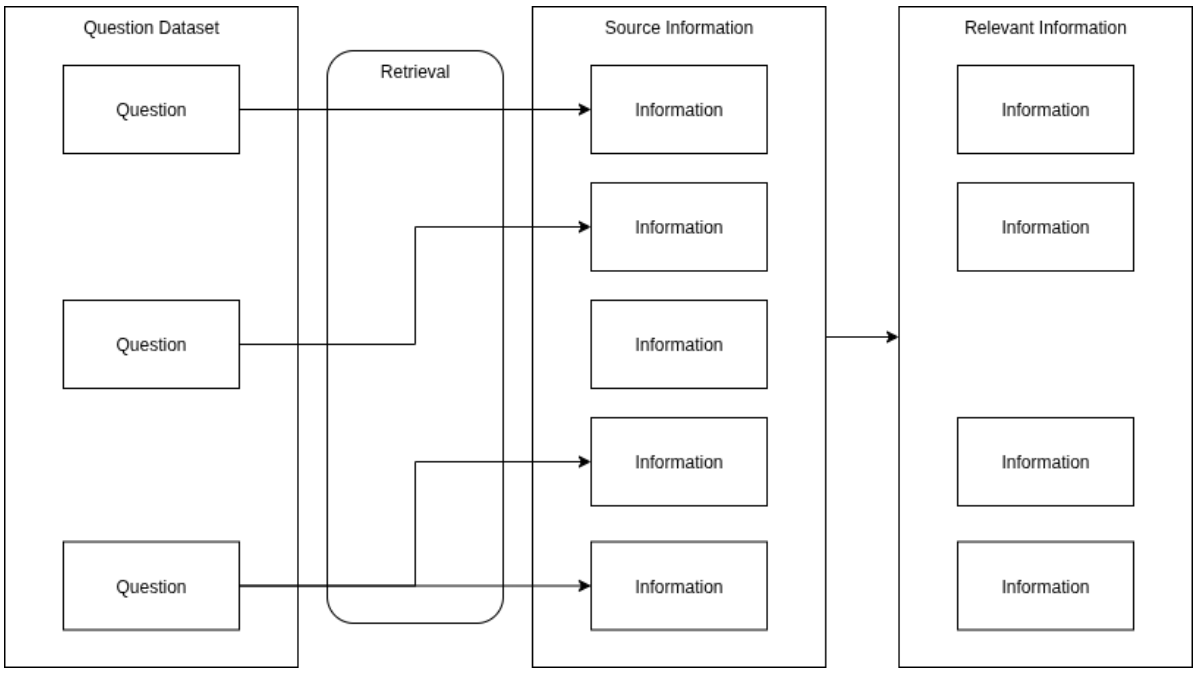

Figure 3.3: Question Based Retrieval

Question based retrieval

Figure 3.3 visualizes the question-based retrieval approach. In case a question data set describing the question domain is available, relevant information from the data source can be extracted using a retrieval approach. To this end, a retrieval pipeline can be employed which splits the text of source documents into chunks, embeds each chunk as well as the question using an embedding model, and computes embedding similarities between each pair of question and text chunk. Alternatively, synthetic questions can be generated for each text chunk, using their embeddings for retrieval. This can help to match relevant questions to text chunks. Based on the resulting similarities, text chunks relevant for the question domain can be separated from irrelevant text chunks. This can be done, for example, by selecting the top k text chunks of each question as relevant pieces of information. Another approach would be to select all text chunks as relevant that exceed a certain threshold of similarity for at least one question.

Interpretable Retrieval

The approach described above uses a standard retrieval method involving an embedding model. Embedding models generate a numerical representation of the semantic content of text. However, the meaning of each element of the vector is not immediately apparent. Similarly, it is not immediately apparent why a text chunk is determined to be

similar and hence relevant for a question. Nevertheless, for each text fragment determined as relevant, one or more responsible questions can be identified. This allows for granular control over the retrieval process as it makes informed manual adjustment of the question data set possible in case irrelevant information was found in the final data pool. Similarly, if crucial information was found to be missing, appropriate questions can be added.

Challenge 2.2: Extraction of key concepts
Identify key objects or concepts in the data source that should be formalized separately as data objects.

An important step in structuring information for the generation of a knowledge base is the identification of relevant concepts and entities within the text. Based on these entities, properties (e.g. in form of facts) and relations (e.g. in form of rules) can be extracted. It can be advantageous to separate these two steps if better control is required over the focus, complexity, and granularity of the final knowledge base. Concepts can gain relevancy by communicating information to the user, by enabling necessary reasoning steps within the system, or by communicating context information of the user to the system.

The problem of key-concept extraction is well known and has been extensively studied. Some promising approaches utilizing LLMs are presented in the following subsection.

LLM based extraction

Ebrahim Norouzi, Sven Hertling, and Harald Sack present an approach to extract key concepts from text using LLMs with specifically crafted system prompts (Norouzi, Hertling, and Sack 2025). A similar approach can be used in this context. Focusing on the extraction of concepts relevant to the question domain and evaluating the three previously mentioned categories of relevance, an LLM can be used to extract key concepts as follows.

For each question category (as described in Section 3.2.1) an LLM is instructed to extract important concepts from the provided source document that are relevant in the context of the question category. A possible system prompt can look as follows:

Prompt 3.4. *Your task is to extract keyphrases from a document that describe the most important topics of the document that are relevant to answer a certain category of questions. Question category: <Question Category Template>. Document: <Document>.*

To guarantee that a well-structured list of categories is returned, the answer of the

LLM can be reformatted as structured output e.g. by using a pydantic schema in combination with function calling capabilities of an LLM. In case a question dataset is available, the resulting list of concepts can be evaluated (ranked and filtered). To this end, for each pair of question-concept pair, the following questions are answered by an LLM:

- Is the concept relevant for communicating an answer to this question to the user?
- Is the concept relevant for reasoning about the result of the question?
- Is the concept relevant for describing context / situations of the user that can be used to answer the question?

As a result, concepts for which all the questions yielded a negative answer can be removed. In addition, concepts can be ranked by the number of questions for which they were determined to be relevant. The approach of extraction, filtering, and ranking enables granular control over the concept extraction process.

Matching MeSH Concepts

- Find important MeSH Concepts in text

Challenge 2.3: Abstraction of concepts
In case relevant concepts within the data source are presented in too much detail, appropriate generalizations and abstractions should be made.

As mentioned in the previous challenge, concepts can be categorized according to the relevance to the user. A concept can be primarily relevant for communicating information to the user, to capture context information of the user, or to enable reasoning processes within the system. Buchanan and Shortliffe formulate this idea as follows: "...[T]he vocabulary that the expert uses to talk about the domain with a novice is probably inadequate for high-performance problem solving." (Buchanan and Shortliffe 1984, p. 150). Although it is possible and not unusual for concepts to fall in more than one category, this perspective helps in placing concepts on the spectrum between user-oriented and system-oriented concepts. An example of a user-oriented concept in the context of medical diagnosis would be the diagnosis itself. It is apparent that the user is interested in being informed about the diagnosis the system provides in an understandable manner, meaning in layman terminology. An example of a system-oriented concept would be thresholds for laboratory tests. Although a user might want to get information about specific methods of diagnosis, this concept is especially relevant

within the system when considering the possibility of a diagnosis. This distinction is of importance when determining the complexity and granularity of concepts. When communicating with the user, concepts should be used at an appropriately abstract level. When reasoning about concepts within the system, as granular and complex concepts as necessary should be used.

This challenge of abstracting concepts is twofold. On one side concepts themselves have to be defined at a reasonable abstraction (e.g. diagnosis, person, disease, ...) on the other side, the concrete terminology with which the categories are instantiated (Typ-2-Diabetes, Diabetes with peripheral circulatory disorders, ...) has to be chosen at a reasonable level of abstraction. A special challenge regarding the terminology in the context of a Prolog knowledge base is that, in the case of user context information, the terminology must not only be similar but identical. If the user provides context information, for example, a symptom, the exact phrasing of the symptom must occur in the knowledge base. This limitation is so strong that a mapping solution between user terminology and knowledge base terminology is necessary not only to communicate through concepts on the same level of abstraction but also to use identical terms within these concepts.

Two central approaches for abstracting concepts can be followed. Either the complexity of concepts themselves is adjusted. This would involve identifying one or more concepts that are too granular and mapping them to a more general concept. The other approach would not change the concepts themselves but would adjust the formulation of answers within the question answering system. Both approaches can be used simultaneously and help adjust the complexity of concepts to their respective purposes.

The approach of adjusting abstraction levels according to concept categories can be of practical use if the use case of the system is clear and the concepts that have to be communicated to the user are limited. Otherwise, answer reformulation should be used because this approach has an important downside: It goes against the declarative nature of expert systems, where knowledge can and should be incorporated without an explicit purpose in mind. This separation of knowledge from an intended purpose allows the knowledge base to cover a large amount of possible question scenarios.

Concept abstraction using LLM

- Use LLM to identify concepts that have to be abstracted (summarized) to be better suitable for communication to the user
- Use LLM to identify concepts that have to be more granular for better reasoning within the system

- Use LLM to find new sets of concepts

Terminology abstraction using ICD10

An important concept within the use case of medical diagnosis that is important to present to the user at a suitable level of abstraction is "diagnosis". If the knowledge base and the user do not share a common vocabulary of terms and phrases to describe diseases and medical conditions, as well as relevant context information such as symptoms and laboratory results, the knowledge base and the user cannot communicate accurately.

Using the example of description of symptoms, the Prolog system might, for example, expect the symptom "susceptibility_to_infections" while the user would choose the term "frequent_infections". While both terms describe the same thing, the knowledge base is not capable of semantically matching the concepts. An additional approach is required to map both phrases to a common terminology. In the medical context, there exist various terminologies that can be used as a basis for a shared terminology between a Prolog Knowledge Base and a user. An example of a candidate for shared terminology is ICD10, which assigns unique identifiers to medical concepts. ICD10 codes and concepts are hierarchically structured so that various levels of details of concepts can be identified. Using ICD10 in this context allows us to solve the challenge in two ways: On one side, it enables the mapping of user context information to an agreed upon terminology. On the other hand, since ICD10 is structured hierarchically, occurring terminology can be abstracted within this hierarchy to more general and understandable terms.

On a technical level, for any given medical term within the knowledge base or provided by the user, the task is to find a semantically similar term contained within the standardized set of ICD10 terms. This task can be described by the following challenges:

- Find candidates of ICD10 terms
- Select a suitable ICD10 term

The most basic approach to solve both challenges at once would be to ask an LLM to answer with an ICD10 code that closely matches a term given within the context of the LLM. This approach relies on the LLM to have memorized the ICD10 database during training. As shown in the evaluation in Chapter 4, this approach is very susceptible to hallucination and does not lead to good results.

Another approach is to build a retrieval pipeline to filter out a small set of relevant ICD10 candidates for a given medical term. The candidate list can then be passed to an LLM with the instruction of selecting the most suitable ICD10 code. The German ICD10 can be downloaded as a CSV file (Bundesinstitut für Arzneimittel und Medizinprodukte

(BfArM) 2025). This file contains ICD10 codes as well as an associated standardized medical term. One line of the file looks as follows:

```
1|99173|1|E10.90||||Diabetes mellitus Typ 1
```

Important for the retrieval system are the ICD10 code "E10.90" and the medical term "Diabetes mellitus Typ 1". To build a retrieval system, both values are filtered for all lines of the file. An embedding model is used to compute a mathematical vector based on the medical term "Diabetes mellitus Typ 1" which encodes the semantic meaning of the term in a vector space. This enables to compute distances between embeddings of medical terms matching their respective semantic similarity. This property of embeddings is exploited in order to find semantically similar terms for a given search term based on a top-k search within the vector space. Once a retrieval system is operational, it can be used in combination with an LLM to automatically determine standardized ICD10 codes for medical terms. Using the retrieval system, the top-k ICD10 terms as well as the corresponding ICD10 codes can be passed to an LLM with the instruction to automatically determine a suitable ICD10 code for a given search term. As ICD10 by nature is a hierarchically structured terminology, for a given search term there might be multiple matching results, which closely match the search term, however, with granular distinctions. In case of the search term "Diabetes Typ 1", a search result might look like:

```
1|115659|1|E10.74||||Typ-1-Diabetes mellitus mit diabetischem Fußsyndrom
1|99128|1|E10.80||||Komplikation bei Typ-1-Diabetes mellitus
1|69653|1|E10.90||||Brittle diabetes
1|99173|1|E10.90||||Diabetes mellitus Typ 1
1|99180|1|E10.90||||Diabetes mellitus Typ 1 beim Erwachsenen
1|99158|1|E10.90||||Diabetes mellitus Typ 1a
1|99159|1|E10.90||||Diabetes mellitus Typ 1b
```

Here, one can see that for the coarse term "Diabetes Typ 1" many more granular terms could be found. Hence, one approach to further standardize the terminology within the Prolog system is to decide to only use the initial three digits of the ICD10 code to cover a more abstract concept. In this example, a suitable ICD10 code that covers all of the results is "E10". This task of determining a suitable three-digit ICD10 code can be added to the LLM instructions. Once a Retrieval System is operational, it can be utilized to standardize the terminology of an existing knowledge base. This task can be described by the following challenges:

1. Find terms to standardize
2. Find suitable ICD10 codes
3. Replace terms in Knowledge Base

The initial task is to find terms in the knowledge base that should be standardized. Recalling that the initial reason for requiring standardization is the interaction with the user, the first approach is to standardize terms used for communication of medical concepts with the user. Looking ahead at solution approaches for challenge 3.2, there exists a predetermined set of predicates for interacting with the user. These predicates (e.g. nutzerangabe_ja(aussage), nutzerangabe_nein(aussage), ...) contain a medical term as an argument of a predicate where the predicate describes the meaning of the medical term within the context of the Knowledge Base. If a fact such as nutzerangabe_ja(bluthochdruck_im_alter) is either located within the Prolog knowledge base or extracted from the user question, standardizing this medical term can help the Prolog system and the user to communicate semantically identical concepts in different terms. Hence, to find suitable terms to standardize, one can use a regular expression searching for all medical terms within the predefined predicates for user interaction.

Answer Reformulation

TODO: Maybe put in QA chapter?

3.2.3 Information Representation

This section presents challenges and solution approaches during the information representation phase.

Challenge 3.1: Utilizing user context information
Make the knowledge base use the context information provided by the user to answer questions.

The goal of an expert system, especially in the case of a medical diagnosis system, is to answer questions concerning not only factual information but also to reason about complex situations. To this end, it is necessary for the user to communicate a situation to the system by providing context information. Regarding challenge 2.2 and 2.3, approaches were described to identify concepts that are relevant for the user to communicate with the system. This challenge focuses on the representation of these concepts within the knowledge base.

Context Facts and Rules

Most information the user provides can be represented as a fact (e.g the user states that he suffers from a symptom). However, the user might also describe a situation which would best be represented as an additional set of rules (e.g. the user states that a symptom occurs only under very specific circumstances). The challenge in representing context information as additional rules is that these rules must be integrated into the existing knowledge base to incorporate them into the reasoning process. This would require an adjustment of the existing knowledge base. It can also occur that the context rules overwrite existing rules, hence changing the behavior of the reasoning process. This can be highly problematic, especially in a medical context, if user queries have the ability to arbitrarily modify the diagnostic process. These problems do not occur if the representation of context information is limited to a set of facts or rules that are determined before creation of the knowledge base and that are explicitly for the purpose of being instantiated with dynamically extracted user information.

Determining Context-User Relation

When representing context information that describes information about the user (either as a fact or a rule), an appropriate relation for the respective concept has to be determined. As part of the solution to challenges 2.2 and 2.3, a set of concepts relevant to the user context are retrieved. This can, e.g. be the concept of "symptom", "laboratory_result", "diagnosis", etc. Based on these concepts, relations to the user that are relevant for the use case have to be found. On possible relation for the concept of "symptom" could be "has_symptom" leading to a fact "has_symptom(Symptom)" describing the fact that a user has a symptom.

To find appropriate relations, the question domain represented by a set of user questions can be utilized (see challenge 1.2). For each question the following approaches can be used to extract relevant relations.

Extracting relations from user questions: The user question usually already contains relevant context information for the question. An LLM can be prompted to extract a set of relevant relations based on the question for a given concept (e.g. symptom).

Retrieval system: As part of challenge 2.1 a retrieval system was employed that collected relevant domain knowledge for each user question of the question domain. For each pair of domain knowledge and question, an LLM can be prompted to extract relations relevant for the question based on the domain knowledge.

Using these approaches, for each concept, a set of relations can be collected. These relation candidates can be postprocessed using an LLM which is instructed to e.g. dedu-

plicate similar relations.

Context integration using LLM prompt

As a result of the previous step, a list of relevant context information representations is available in the form of relations (e.g. "has_symptom(Symptom)", "positive_laboratory_test(LaboratoryTest)", ...). The goal is to enforce the Prolog knowledge base to utilize these facts within the reasoning process when necessary. This can be done by formulating a suitable LLM prompt containing all relevant relations and instructing the LLM to incorporate them during the creation of the Prolog knowledge base. It is important to hint to the LLM, that the relations will be added dynamically and will contain (possibly unreliable) user information either in form of facts or rules.

Obtaining context information

Once representations of context information are found and incorporated into the knowledge base, the important task is to instantiate the context information with concrete values based on the information provided by the user.

There are two methods for obtaining context information from the user: Either the information is extracted from natural language (e.g. from the initial question of the user), or information is directly queried by the system in a structured manner.

Extraction from natural language: As for the extraction of context information from natural language, a solution has to be found to identify context information and translate it into the available context representations. If the text containing the context and the number of context representations is sufficiently small, an LLM can be tasked to solve the extraction. To this end, the text as well as all available context representations (formulated as a Prolog head, e.g. "has_symptom(Symptom)") can be added to the context of an LLM along with the instruction to extract all context information using the available context representations. An example prompt can look as follows:

Prompt 3.5. *Your task is to represent all context information provided by the user utilizing as Prolog facts by utilizing only the following Prolog heads. User text: <Context Information> Prolog heads: <Prolog Heads>.*

If the context becomes too large, retrieval approaches can be employed. To this end, all available context representations have to be described in natural language which is then embedded using an appropriate embedding model. The textual description for a context representation can be generated using an LLM with the instruction to generate a natural language example of information that can be represented by the given Prolog

head. With the embeddings for the context representations available, for each user sentence, candidates of context representations can be retrieved. Using the same LLM prompt as previously but including only the most relevant candidates as well as the user sentence in the LLM prompt reduces the context length a lot. As a result, a list of valid Prolog facts is generated which can be added to the bottom of a Prolog knowledge base to incorporate user context into the reasoning process.

Context queries:

It can not be expected that a user is aware of all the context information required by a system to solve a reasoning problem. Hence, it is necessary for the system to dynamically ask the user for specific information when it becomes relevant. As described in a previous subsection, context representations are integrated into the Prolog knowledge base. This means that as part of a Prolog rule during the reasoning process, contextual information such as "has_symptom(Symptom)" is evaluated. If this information is already available as part of the extraction process from e.g. a user question, the user must not be asked for this information. If the user, however, did not specify this information, he should be prompted a yes/no question. This can be realized in Prolog by adding a rule capable of querying the user for information if it is not available. Such a rule can look as follows:

```
nutzerangabe_ja(Aussage) :-
    nutzerangabe_ja_gespeichert(Aussage).

% Frage Nutzer nach einer Angabe und speichere die Antwort
% falls diese "ja" ist. Andernfalls speichere "nein".
nutzerangabe_ja(Aussage) :-
    \+ nutzerangabe_ja_gespeichert(Aussage),
    (   nutzerangabe_ja_impl(Aussage)
    ->  assertz(nutzerangabe_ja_gespeichert(Aussage)), true
    ;   assertz(nutzerangabe_nein_gespeichert(Aussage)), false).
```

This Prolog code consists of two rules. The first rule is evaluated first and checks if certain context information has already been provided. "nutzerangabe_ja_gespeichert(Aussage)" checks if context information e.g. "nutzerangabe_ja_gespeichert(has_symptom(migraine))" already exists as a fact. The second rule is evaluated after the first rule. In case no fact "nutzerangabe_ja_gespeichert(Aussage)" exists, using "nutzerangabe_ja_impl(Aussage)", the user is prompted to provide the information by answering a question either with "yes" or "no". The expression evaluates to true if the user answers with "yes" and false otherwise. Hence, according to the answer, the fact is stored with either a positive answer "assertz(nutzerangabe_ja_gespeichert(Aussage))" or a negative one. The result of

these two rules is that the user can interactively be queried to answer questions once the information is needed during the reasoning process while saving the already answered questions.

TODO: Describe in more detail

Challenge 3.2: Structuring of knowledge base at meta-level
Realize a formalism within a knowledge base that describes and structures the knowledge at the meta-level. This structure can, among other things, simplify the extension by additional knowledge of similar structure by embedding it into existing meta-structures.

TODO: explain why this is even necessary
TODO: add referene (chapter 09 MYCIN)
TODO: describe approaches

- Prepare a meta-level prolog program by hand and add it into LLM context (e.g. see Prolog Template approach)
- Let LLM generate a meta-level Program with an appropriate prompt

Challenge 3.3: Formalization of knowledge in Prolog
What are the approaches to formalizing information using Prolog?

TODO: formalization or representation or both?
TODO: formalization happens before representation in prolog!
TODO: Formulate as text:

- Prompt LLM to translate a bunch of information into Prolog at once
- Prompt LLM iteratively to translate individual pieces of information into Prolog while having the whole KB in Context
- Use Chain-of-Thought Prompt instructing LLM to solve question as a reasoning process using Prolog (integrate new Prolog code into KB as second step)
- Use teacher-student approach: RAG system is teacher asking the Prolog KB questions. Prolog KB answers with result + reasoning and RAG system instructs corrections if the answer or reasoning is faulty

TODO: explain advantages of iterative KB building approaches => possibility of continuous extension of KB
TODO: relates to section 3.3.6, maybe integrate it here?

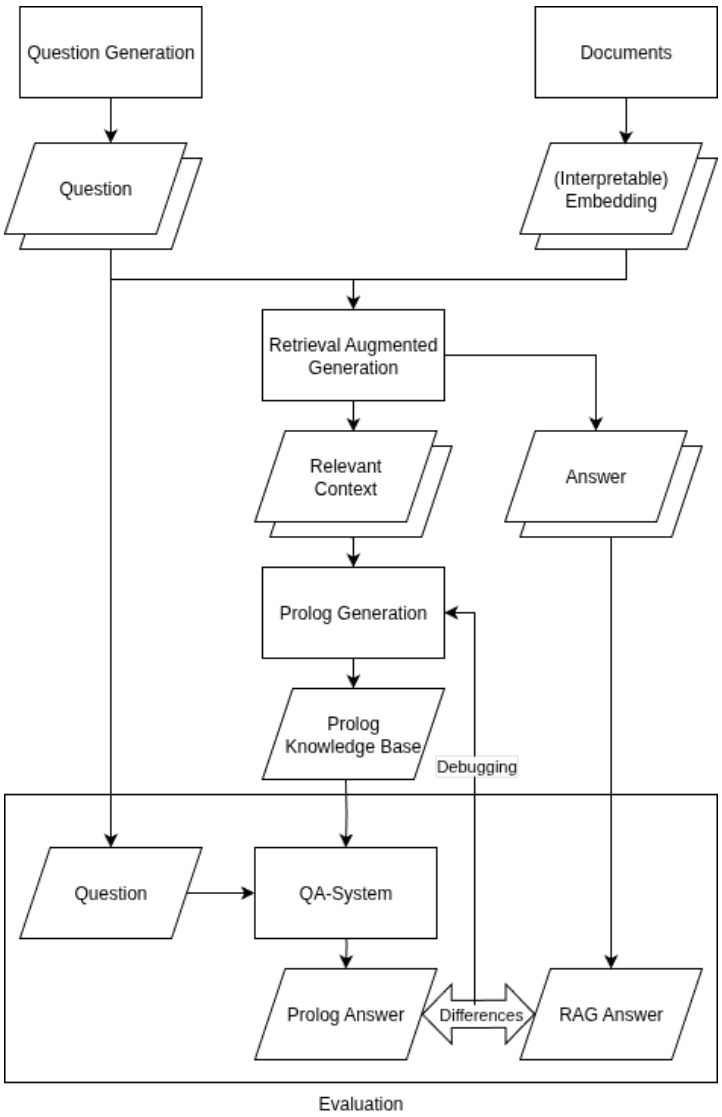

Figure 3.4: Knowledge Acquisition Technical Pipeline

3.2.4 Design Variants

Figure 3.4 shows the technical pipeline of the knowledge acquisition task using the approaches described in Section **??**. The following subsections go into detail on concrete design variants of this pipeline. Beginning with a very basic solution, each subsequent variant becomes more advanced, adding more approaches described in the knowledge acquisition pipeline. The variants described in this chapter are used for evaluation in Chapter 4.

Basic Prompting

The first approach is a baseline that aims to implement the previously described data processing steps all at once using a singular LLM prompt.

Prompt 3.6. *(Knowledge Base Extraction Prompt): You are an expert in the creation of Prolog Knowledge Bases. You receive a medical text document. Your task is to develop a Prolog Knowledge Base that contains all relevant facts and rules of the medical document that are relevant for the following use case: "Medical diagnosis"*

In addition to this instruction, the whole text of the medical document is included within the context of the model. In case of multi-modal models, images can be added into context. Otherwise, a separate vision model can be used to create a textual explanation of each image, which is then inserted into the document at the corresponding place.

Based on this prompt, the model is expected to perform the aforementioned data processing steps at once, directly extracting a Prolog Knowledge Base from the medical text. Implicitly, the model has to recognize all information from the text that is relevant for the specified use case and reformulate it into respective facts and rules in Prolog.

One limitation of this approach is that all medical text has to fit into the context of the model. If the medical text is very large, this might not be possible or might lead to degraded quality of the output. Methods for solving this issue are covered in Section 3.3.6.

Another issue of this approach is that the specification of the use case remains very abstract. It is possible to add more paragraphs to the prompt detailing on the exact use case at hand. This provides the model with more information to distinguish relevant information from irrelevant information.

Prompting with Template

One approach for solving the unstable generation of Prolog Knowledge Bases is to give the LLM an initial handcrafted version of a Knowledge Base which contains generally applicable rules and facts that are relevant to the given use case. Here, the system designer can use domain knowledge to craft an initial Prolog Knowledge Base. This Prolog Knowledge Base, which can be called a Template, has two main advantages: By carefully designing the template, the system designer can guide the LLM to focus on specified concepts while enriching them with case-specific information retrieved from the medical text. This can help to reduce the variance between Prolog Knowledge Base Extractions and also to specify a targeted level of abstraction regarding medical concepts. Another main advantage is that a template provides a minimal set of rules

and facts to which all Prolog Extractions adhere. As a result, Prolog Knowledge Bases extracted separately e.g. for different kinds of disease can be utilized by accessing information through the rules and facts specified as part of the template.

A minimal Prolog template for the scenario of medical diagnosis might look as follows:

```
% Allgemeine Regeln fuer die Diagnose von Krankheiten

% Krankheit ist diagnostiziert, wenn sie durch
% eine Diagnostik diagnostizierbar ist und das
% Ergebnis der Diagnostik positiv ist
krankheit_ist_diagnostiziert(Krankheit) :-
    krankheit_diagnostizierbar_durch(Krankheit, Diagnostik),
    diagnose_ergebnis(Krankheit, Diagnostik, positiv).

% Das Ergebnis einer Diagnostik kann das
% Ergebnis eines Tests sein
diagnose_ergebnis(Krankheit, Diagnostik, Ergebnis) :-
    test_ergebnis(Krankheit, Diagnostik, Ergebnis).

% Eine Krankheit ist diagnostiziert, wenn sie durch Symptome
% diagnostizierbar ist und alle Symptome vorhanden sind
krankheit_ist_diagnostiziert(Krankheit) :-
    krankheit_diagnostizierbar_durch_sympt(Krankheit, Sympt),
    symptome_vorhanden(Sympt, positiv).

% Eine Diagnostik ist indiziert fuer eine Krankheit, wenn die
% Krankheit durch eine Diagnostik diagnostizierbar ist und
% eine Liste von Symptomen existiert, die auf eine Krankheit
% hinweist und alle Symptome vorhanden sind
diagnostik_indiziert_durch_sympt(Krankheit, Diagnostik) :-
    krankheit_diagnostizierbar_durch(Krankheit, Diagnostik),
    krankheit_diagnostizierbar_durch_sympt(Krankheit, Sympt),
    symptome_vorhanden(Sympt, positiv).
```

The declarative nature of Prolog enables an intuitive way for a system designer to define rules and facts on an abstract level while leaving the more granular and case-specific declarations open for the LLM. In this case the LLM would be required to define disease-specific facts and rules such as: "krankheit_diagnostizierbar_durch(diabetes, blutzuck-

ertest)" and "krankheit_diagnostizierbar_durch_sympt(diabetes, infektionsanfälligkeit)". This approach also paves the way for leveraging patient-specific information by defining the fact "symptome_vorhanden(Sympt, positiv)" for which the required information can, for example, be retrieved using Prolog features such as the "read" and "assertz" predicates.

The main disadvantage of this approach is that a system designer is required to manually implement the Prolog template. This is an issue because it is difficult to construct a Prolog Template which on one side is general enough to be applicable to all relevant use cases but on the other side also extensive enough to cover all important concepts of the domain.

This issue can already be observed in the example above: The rules defined for diagnosing a disease are so general, that they can not reflect most real life scenarios. For example, the rule for diagnosing a disease based on a symptom requires that only one symptom be observed that matches the disease. In many cases, this would hardly be recognized as a plausible diagnosis. Hence, case-specific rules for each disease might be required to accurately formalize their respective requirements for diagnosis.

Prompting with System Requirements

The main problem in the approach above is that a system designer is required to develop a reasonable Prolog Template based on sufficient domain knowledge. This becomes especially difficult if the required rules and facts are difficult to generalize and should rather be defined on a case-to-case basis.

An alternative but similar approach is not to implement a Prolog template, but to define system requirements. In the context of Prolog this can, for example, be Prolog headers (predicate + arguments) which describe which user queries as well as which user information the system is expected to be able to process. The main difference to the approach above is that no valid Prolog Knowledge Base is created but only a list of Prolog headers without implementation is defined. The task of the LLM during extraction of the Knowledge Base is to implement these Prolog headers for the context at hand. This can be done for matching contexts (e.g. diseases) in parallel.

Knowing that an implementation of this list of Prolog headers is available for each context, they can act as a "contract" between the Knowledge Base and the rest of the system, making the Knowledge Base reliable to interact with.

Following this approach, on one side, a system designer is able to define requirements of the Prolog Knowledge Base in a simple and descriptive fashion while leaving the more complex and case specific implementations to the LLM.

One example of defining such requirements can look as follows:

```
Angaben:
- diagnose_verdacht(Krankheit)
- diagnose_abgesichert(Krankheit)
- test_empfohlen(Krankheit, Test)
- diagnostik(Krankheit, Diagnostik)

Anfragen:
- nutzerangabe_ja(aussage)
- nutzerangabe_nein(aussage)
- nutzerangabe_wert_ist_groesser_als(name, wert)
- nutzerangabe_wert_ist_kleiner_als(name, wert)
```

The above example shows how simple yet effective requirements can be defined. The system designer lists some of the most relevant pieces of information a user might ask about which are the diagnosis to a disease, recommendation for a medical test as well as suitable methods of diagnosis for a disease. This list can easily be extended. The LLM tasked with the implementation of these requirements receives crucial information about the targeted focus of the system while having the freedom of adapting its implementation to case specific requirements originating form the medical text.

Another important advantage is that the system is enabled to utilize a given set of predicates for accessing user information. This way of standardizing the user input opens many possibilities for designing a framework of interaction between the user and the Prolog Knowledge Base.

Retrieval Based Filtering

Interpretable Embeddings

3.2.5 Possible Extensions

> **Challenge 4.1: Debugging the knowledge base**
> Find and repair errors in the knowledge base that occur due to a failure in knowledge acquisition.

TODO: mention all other possible sources of errors and think about whether to handle them or not

The following subsections leverage the solution approaches presented above in order to present a set of solution packages for the purpose of implementation and evaluation.

NOTE: The following subsections were written at an early stage and have not yet been re-worked. They will eventually contain and reference the approaches and challenges mentioned above.

Challenge 4.2: Extending the knowledge base
Add new knowledge to the knowledge base.

TODO: "After initial design and prototype implementation, the system grows incrementally both in breath and depth" (MYCIN, Chapter 7, 150)

Challenge 4.3: Definition of test cases
Add automated test cases that identify detrimental effects of modifications of the knowledge base.

TODO: helps with modularizing KB / making it more declarative

Challenge 4.3: Rule statistics
Log statistics of rules (e.g. did a rule ever evaluate to true?) TODO: See MYCIN Chapter 7, page 157

Challenge 4.3: Contradiction Identification
Identify contradicting rules. TODO: See MYCIN Chapter 7, page 157

3.3 The Request Processing Pipeline

This section describes the Request Processing Pipeline which is responsible for answering user questions by evaluating them within an existing Prolog Knowledge Base. The main task of the pipeline is to realize a convenient way for the user and the Prolog Knowledge Base (being executed within a Prolog runtime) to exchange information and instructions. The user initiates the interaction by asking a question in natural language. This question has to be reformulated into a Prolog query such that it can be executed within a Prolog runtime. It is important, that the query closely matches the intent of the question. Otherwise the Prolog system computes a possibly correct answer for a different question. The answer computed by the Prolog system consists of truth values or variable instantiations. Hence, for user convenience, Prolog answers should be translated back to natural language.

3.3.1 Mapping Questions to Prolog

As it is prohibitive for a user to formulate syntactically correct Prolog queries that can be directly executed within the Prolog runtime, another communication method needs to be used.

As Prolog is a declarative programming language by nature and operates on humanly understandable concepts, one could implement a user interface which directly visualizes existing predicates as well as their meaning. Users could then search and select predicates of interest and either declare them as new facts or use them to build up a query. This approach would rely on a intuitive and self-explanatory user interface which reduces the complexity of the Prolog programming language to a minimal amount required for constructing queries.

Another approach would be to enable the user to formulate questions in natural language. This has the potential to enable for a more seamless interaction of the user with the system. However, it brings up the challenge of mapping user questions to suitable Prolog queries which can be described with the following challenges:

1. Evaluate if the question can be answered by the existing Knowledge Base
2. Find all relevant predicates in the Knowledge Base
3. Construct a Prolog query that is semantically identical to the user question

The simplest approach to solve all three challenges at once is the usage of a single LLM prompt which receives the whole Knowledge Base as context and is instructed to create a suitable Prolog query if possible or to state if it is not possible. To make it easy to distinguish between questions that can be translated to Prolog queries and questions which are not translatable, the LLM can be forced to respond in a predetermined json format, with one of the json values indicating whether a translation is possible or not.

This approach relies on the Knowledge Base being sufficiently small to handle within a LLM context at once. If this is not the case, other strategies can be employed. One option could be to index individual Prolog rules and facts of the Knowledge Base using automatically generated descriptions. This can result in a retrieval system capable of determining the relevant Prolog rules and facts for a given user question. An LLM can then be prompted similar to the approach above, however, only with relevant parts of the Knowledge Base occurring in the context.

TODO: Additional approaches for verifying, that the question and query match semantically (maybe add description for each predicate to better match them to user questions?)

3.3.2 Extracting Facts

Following the approach of designing a user interface that enables the user to directly construct Prolog queries based on predicates existing in the Knowledge Base as described in Subsection 3.3.1, an additional feature can be added not only to create questions but also to specify facts relevant for a stated question before execution. This way, the user can be made aware of available and relevant concepts within the Knowledge Base and can be guided to define respective values for the given situation.

If natural language questions are used as described in Section 3.3.1, the question can directly contain important information about the user that is relevant to answering the question. A basic example would be that the user asks for a medical diagnosis while directly providing a list of experienced symptoms. Approaches are required for extracting this information.

A basic approach would be similar to 3.3.1. While the task is not to construct Prolog queries but Prolog facts, the challenges remain the same. Hence similar approaches using a LLM or a retrieval system can be employed

Once relevant information has been extracted in form of valid Prolog facts, they can be either temporarily or persistently appended to the existing Knowledge Base. This way, once a query is executed on the extended Knowledge Base, the newly added facts are taken into consideration during the evaluation.

3.3.3 Executing Queries

TODO: Handling of closed world assumption

Once a valid Prolog query has been generated, and possible user facts have been appended, the query has to be executed on the Knowledge Base. Many implementations of a runtime for the Prolog programming language exist, one of the most famous being SWI-Prolog which was used in this thesis. In order to seamlessly integrate the usage of a Prolog runtime within Python, a tool called PySwip can be used. This tool reduces the execution of Prolog queries to simple Python function calls and also provides for deeper integration between both programming languages as will be discussed in Subsection 3.3.5. The process of executing a Prolog query is straight forward and can be done by loading the Knowledge Base from the file system and then executing the query string. The result of the query is returned as a Python dictionary consisting of an array of objects representing the variations of paths for which the query evaluated to "True".

TODO: Explain this in more detail e.g. in the Realted Work section?

In case the Prolog query contains variables, the Prolog runtime answers with possible values for the variables for which the query evaluates to true.

3.3.4 Natural Answer Generation

As previously described, the Prolog runtime answers with a structured format containing the truth values as well as possible evaluation of variables. For the convenience of the user, this response can now be translated into a natural language answer.

A basic approach can be the usage of a LLM which receives the structured output of the Prolog runtime as well as the original user question along with a prompt instructing the LLM to generate a suitable answer to the question.

Depending on the use case a variety of phrases can be specified for the LLM to adhere to, to hint the underlying operation of the system to the user. One important information for example would be, that as a result of the evaluation of the question within the Prolog system, the answer "no" to a question must not mean, that the answer is false under any circumstances. It rather means, that the question of the user could not be determined to be true based on the information encoded within the Knowledge Base. It might as well be the case, that the Knowledge Base lacks the necessary information to correctly identify the question as "true". As this distinction is not apparent to the user, it is advisable to incorporate such information in the answer of the LLM.

3.3.5 User Interaction

As described in Subsection 3.3.2, information about the user can be extracted from the initial question. However, it is also important to dynamically receive relevant information about the user during execution time. In this way, the Prolog system has the ability to explore all possible user-specific cases that might lead to an answer to the question.

As indicated in Subsection 3.3.3 the Python tool "PySwip" can be used to achieve this by registering Prolog predicates as Python functions. Once a registered predicate is evaluated during the execution of a query, the corresponding arguments are passed to the matching Python function which is executed. The implementation of the Python function can be implemented arbitrarily must return either the value True or False which is returned to the Prolog runtime as truth value of the corresponding predicate.

Leveraging this integration possibility, dynamic user input can be realized by registering Prolog predicates such as nutzerangabe_ja(Aussage), nutzerangabe_nein(Aussage), messwert_groeßer_als(Messwert, Grenzwert). The respective Python implementations can contain a standard "input" call that receives user feedback and determines the truth value of the fact depending on the user input. The Python function then returns either True or False depending on the user input which is then passed to the Prolog runtime as truth value of the corresponding predicate.

TODO: explain how to save facts instead of asking the user the same question multiple

times

In order not to ask the user the same question multiple times, each question, once answered, can be saved as a Prolog fact.

3.3.6 Explainability of Reasoning

Visualizing the Reasoning Process

Visualizing the Knowledge Base

4 Experimental Evaluation

4.1 Evaluation of the Knowledge Acquisition Pipeline

Sources of errors in Knowledge Base creation (MYCIN, Chapter8, p. 159,160)

- Source of knowledge: incomplete, erroneous, outdated
- Selection of knowledge: removal of important information
- Formalization of knowledge: syntax errors, content errors

4.1.1 Comparison of RAG and Prolog Answers

4.1.2 Knowledge Base Evaluation

1. manually or automatically created test cases

TODO: Metrics: (See MYCIN, Chapter 08, p. 161ff.)

- Logical Checks for Consistency (Conflict, Redundancy, Subsumption)
- Logical Checks for Completeness (Missing rules)

4.2 Evaluation of the Question Answering Pipeline

5 Conclusion and Future Work

5.1 Future Work

5.1.1 LLMs as medical expert in interactive expertise transfer

e.g. ROGET

5.1.2 Alternative Domains and Applications

- Document specific KB in legal context (e.g. querying of legal contracts)

Bibliography

Buchanan, Bruce and Edward Shortliffe (Jan. 1984). *Rule-based Expert System – The MYCIN Experiments of the Stanford Heuristic Programming Project*.

Bundesinstitut für Arzneimittel und Medizinprodukte (BfArM), ed. (2025). *ICD-10-GM 2026 Vorabfassung Überleitung zwischen Version 2025 und 2026 TXT (CSV). Stand: 30. Juli 2025*. Im Auftrag des Bundesministeriums für Gesundheit (BMG) unter Beteiligung der Arbeitsgruppe ICD des Kuratoriums für Fragen der Klassifikation im Gesundheitswesen (KKG). Köln. URL: https://www.bfarm.de/DE/Kodiersysteme/Services/Downloads/_node.html (visited on 07/30/2025).

Chen, Yanda et al. (2025). *Reasoning Models Don't Always Say What They Think*. arXiv: 2505.05410 [cs.CL]. URL: https://arxiv.org/abs/2505.05410.

Jr, Claudionor Coelho, Yanen Li, and Philip Tee (2025). *Do LLMs Dream of Discrete Algorithms?* arXiv: 2506.23408 [cs.LG]. URL: https://arxiv.org/abs/2506.23408.

Norouzi, Ebrahim, Sven Hertling, and Harald Sack (2025). *ConExion: Concept Extraction with Large Language Models*. arXiv: 2504.12915 [cs.CL]. URL: https://arxiv.org/abs/2504.12915.

Rogers, Anna, Matt Gardner, and Isabelle Augenstein (2021). "QA Dataset Explosion: A Taxonomy of NLP Resources for Question Answering and Reading Comprehension". In: *CoRR* abs/2107.12708. arXiv: 2107.12708. URL: https://arxiv.org/abs/2107.12708.