Utilizing Large Language Models for Causal Discovery and Legal Text Interpretation: A Case Study on the German GoZ

Jonas Zausinger
Technical University of Munich
jonas.zausinger@tum.de

Abstract

This work investigates the application of Large Language Models (LLMs) for extracting causal relationships and enhancing reasoning abilities in the context of legal texts, with a particular focus on assessing the legal correctness of contracts. First, the causal discovery capabilities of LLMs are examined and improved using retrieval-augmented generation techniques. The approach is then adapted to legal texts, translating them into logical formulations to facilitate the automated assessment of contract correctness, such as in invoices. As a case study, the correctness of dental invoices is examined based on the German Dentist Fee Schedule (GoZ). The results show that our method has an increased accuracy for contract validation compared to conventional retrieval augmented LLMs.

1. Introduction

Reasoning ability, particularly in extracting causal relationships, is critical for a plethora of applications, including scientific discoveries and legal interpretation. With the evolution of LLMs, there arises an opportunity to harness these models for advanced tasks, such as finding causal relationships or automatic reasoning from legal texts [15].

The central question this paper addresses is the feasibility of using LLMs to extract causal relationships and knowledge from legal texts with such efficacy that it is possible to automatically reason with it and automatically determine the legal correctness of a contract according to the rules of the text.

In the first part of this paper, we examine the ability of LLMs to perform causal reasoning. The ability of LLMs to perform causal reasoning has already been heavily studied. Kiciman *et al.* [15] found that LLMs are able to outperform previous state-of-the-art algorithms in this discipline. LLMs, unlike data-driven state of the art algorithms, do not require vast amounts of numerical data in which causal relationships become visible, but can infer the correct causal relationship based on their knowledge of the world [32]. We build on the work of Kiciman *et al.* [15] by improving the causal reasoning capabilities of LLMs, through the use of retrieval augmented language models.

In transitioning to the second part of the paper, we apply the insights and parts of the methodologies developed in the first part to the domain of legal texts. Traditional machine reading models, even when fine-tuned for legal contexts, exhibit low out-of-the-box performance on legal reasoning due to the unique nature of legal texts, where essential information (such as laws) is often stated only once and differs from the distributional language statistics typically used in machine reading methods [11]. In addition, we cannot adopt our approach from the first part directly, as we are now dealing with complex legal texts and not just single causal variables. So to extract causal knowledge from the legal text we adapt our approach for the legal domain by translating legal texts into logical formulations. This method is informed by both past successful manual translations of legal texts into logical formats [25] [26] [27] [28] [29] [11] [12] and research demonstrating the enhanced problem-solving ability of LLMs when translating natural language problems into symbolic formulations, followed by symbolic solver inference [10] [22] [31]. With this approach, the causal relationships contained in the text become inherently encoded in the logic formulations. Hereby we reuse our insights from the first part, according to which the performance of causal recognition improves with more contextual information. We can apply this to the translation step from legal text to logic language, as this is also a kind of causal discovery step, where causal relationships expressed in natural language are converted into logical formulations. We chose Prolog as logic language due to its established use in scientific publications [10] [22] [31] and its proven capability to express and apply legislation [25] [26] [27] [28] [29] [11] [12]. Prolog is particularly well suited to the processing and inference of legal texts, because of its declarative nature, use of predicate calculus and efficient coding [5].

Having the legal text translated as a logic language, we

then use this translation to automatically conclude whether a contract based on this legal text is valid or contains errors.

2. Related Work

2.1. Causal Discovery with LLMs

The ability to do causal discovery has profound consequences in areas of societal importance like medicine, science, law, and policy-making, as identifying causal links, as opposed to mere correlations, enables policymakers, scientists, and businesses to make more informed decisions [3]. Identifying what causes certain outcomes can help in risk assessment and preventive measures [23]. In healthcare, understanding the causal mechanisms of diseases can lead to personalized treatment plans and precision medicine, improving patient outcomes [9].
Kiciman *et al.* [15] investigate the ability of LLMs to perform Causal Reasoning in many different domains. Among other things, they test the ability of LLMs on Pairwise Causal Discovery tasks. Typically, these tasks are set to discover causal relationships between two variables based on observational data. In this work, they do not rely on observed data, but directly ask large language models whether one variable causes another variable. They test this on the Tübingen cause-effect pair dataset [19].

2.1.1 Pairwise Causal Discover on the Tübingen cause-effect pairs dataset

This dataset [19] consists of 108 different cause-effect pairs from various fields, including meteorology, biology, medicine, engineering, and economics. Since, to learn a causal graph, a fundamental ingredient is to find the causal direction of two variables, it is a widely used dataset to benchmark causal discovery algorithms. The best known accuracy is 83 % [15], achieved by the Mosaic algorithm [30], which is a traditional data-driven discovery algorithm. Kiciman *et al.* [15] now simply asks an LLM whether one variable causes another variable. In doing so, they achieve 89 % accuracy with gpt-3.5-turbo and 96 % with gpt-4.
In addition to pairwise causal discovery, they also investigate the discovery of a full causal graph.

2.1.2 Full graph discovery on the arctic sea dataset

In full graph discovery, one has a set of variables and wants to determine for all possible pairs whether there is a causal relationship, and if so, which of the variables is the cause and which the effect. For simple graphs over 3-4 variables, Long *et al.* [18] show that LLMs can obtain promising accuracy in inferring the edges. Kiciman *et al.* [15] expand the problem to more complex, real-world datasets involving specialised knowledge. One of them is the arctic sea ice coverage dataset [13]. This scientific dataset is about

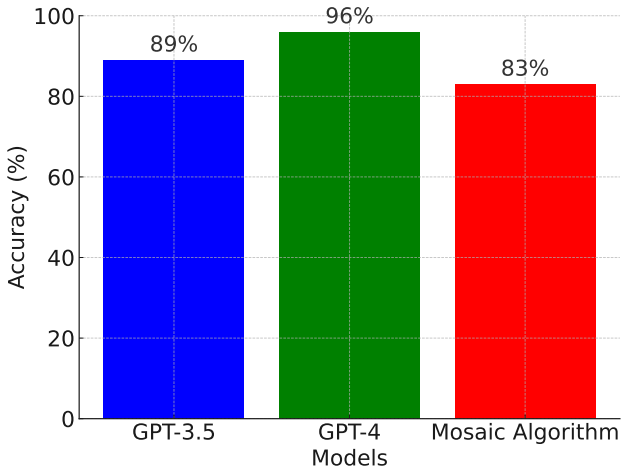

Figure 1. Pairwise Causal Discovery Accuracy on the Tübingen Dataset. The LLM-based causal discovery algorithms beat the data-driven state of the art discovery algorithm Mosaik.

the drivers of arctic sea ice coverage, what causes the arctic sea ice coverage to increase or decrease. The ground truth Causal Graph Huang *et al.* [13] constructed from domain-knowledge has 12 variables with 48 edges. Huang *et al.* [13] also evaluate the accuracy of data driven causal discovery algorithms and evaluate their effectiveness through the normalized hamming distance (NHD) between the predicted graph G' and the ground-truth graph G. It is a widely used metric to compare two unweighted graphs on the same set of vertices [8]. For a graph with m nodes, this is

$$NHD(G, G') = \frac{1}{m^2} \sum_{i,j=1}^{m} 1_{G_{ij} \neq G'_{ij}}$$

Since the NHD depends heavily on the number of edges returned by a discovery algorithm, it is not a sufficiently meaningful metric on its own. Therefore, in addition to Accuracy, F1 Score and NHD, they also measure the NHD Ratio. The NHD ratio is the ratio between the NHD of the discovery algorithm and a "floor" baseline that outputs the same number of edges, but of them are incorrect. A lower ratio implies the multiple by which the graph discovery algorithm is better than the worst baseline returning. Kiciman *et al.* [15] obtain a normalized hamming distance of 0.33, comparable to the three recently proposed covariance-based causal discovery algorithms. However, the LLM returns 62 edges, so its NHD ratio (0.43) is substantially better than covariance based algorithms, indicating the competitiveness of LLM-based algorithms for causal discovery.
In our work, we first replicate the results of Kiciman *et al.* [15] on these two datasets and try to improve the causal discovery capabilities of LLMs by performing retrieval-based augmentation with a knowledge base.

2.2. Automatic Reasoning with Legal Text

Automated understanding of legal texts can revolutionize legal research, contract analysis, and compliance monitoring, making these processes more efficient. Opening the ability to quickly parse through vast amounts of legal documentation, identify relevant information or automatically checking contracts could massively reduce the workload of legal professionals. One could for example imagine a software system that will be able to swiftly sift through thousands of pages of legal documentation from various sources. This system could identify key clauses, assess compliance and highlight potential legal issues, all within minutes. This makes automatic reasoning with legal texts a broad and interesting field in research. There are two main areas of existing research that are relevant to us. On the one hand, there are those who try to understand legal texts by means of neural networks and to get answers to legal questions from them [20], [24]. But straightforward application of machine reading models exhibit low out-of-the-box performance, due to difference between legal texts and most existing work in machine reading, which is that much of the information needed for deciding a case is declared exactly once (a law), whereas the information needed in most machine reading methods tends to be learned through distributional language statistics [11].
On the other hand, there are those who try to manually or automatically convert legal text into a form of logic. The disadvantage of this approach is of course the manual effort, the advantage is that one don't has to rely on the legal knowledge being correctly coded in the weights of the network, but has a verifiable logical programme where one only has to query a logic solver for the answer. There have been several efforts to manually translate law statutes into expert systems. Sergot *et al.* [28] translate the major part of the British Nationality Act 1981 into around 150 rules in Prolog, proving the suitability of Prolog logic to express and apply legislation. In addition, there are several other translations of legal text into Prolog or dialects of Prolog, such as the translation of Canada's Income Tax Act into Prolog [29], the translation of the Japanese Civil Code into a Prolog dialect [26], [27], [25], and the translation of the US tax code into Prolog [11]. As mentioned in the introduction, our approach is to automatically translate the legal text into the logical programming language Prolog with the help of an LLM.

2.2.1 Prolog

The reason that both in our work and in many existing research Prolog was chosen as the logical programming language for legal text is due to its characteristics. Legal texts are often comprised of facts and rules. Prolog's declarative nature means that these can be easily encoded into a Prolog knowledge base [26].
Also legal arguments are often structured logically, involving premises leading to a conclusion. Predicate calculus is inherently suited for representing such logical structures, making Prolog an ideal tool for modeling legal arguments [26].
Prolog allows users to query both the explicitly stated information and the implications of this information. This is corresponds to how a legal professional might extract information or interpretations from a legal document [28]. Furthermore in legal texts, much of the reasoning involves deriving implications from established facts. Prolog's ability to handle such inferences aligns well with this aspect of legal reasoning [5].
In Prolog rules are represented as Horn clauses. A Horn clause is a disjunctive clause (a disjunction of literals) with at most one positive, i.e. unnegated, literal [6]. In Prolog syntax this is written as $A :- B_1, \ldots, B_n$. Every Horn clause has exactly one conclusion A and zero or more conditions B_i. The conclusion A is satisfied if and only if all conditions B_i are satisfied. Horn clauses are invoked or queried by a query that is a Horn clause without a conclusion. Horn clauses without conditions are called facts because they are always satisfied. Any set of definite Horn clauses is a Prolog program [26]. Since Prolog has the above mentioned characteristics that make it a very suitable language for legal reasoning, and it is already well established in this domain, as shown by several examples [28] [29] [26] [27] [25] [11] [12], we chose also to use Prolog in this work.

2.2.2 Automatic Rule Extraction and Translation

There are different approaches to translate rules automatically. For example, Boufrida *et al.* [4] use various natural language processing tools and text mining techniques to translate rules in a medical text from a given Ontology Web Language (OWL) into logical Semantic Web Rule Language (SWRL) rules. They use classical NLP techniques (no LLMs). The disadvantage of their approach is that an ontology must already exist.
Other research focuses on initialising a given logical program from a natural language text by parsing the facts from the text into the logical language using Deep Learning models [21], [33]. With this approach, however, the logical programme must already exist which is not the case with our problem.
Our approach differs from those mentioned in that we have only given the basic structure of an invoice. Using this structure, we want to automatically extract all rules from the legal text and translate them into Prolog, which deal with the correctness of invoices. Afterwards, we want to automatically check the correctness of invoices that are already

in JSON format.

2.2.3 Reasoning with Prolog

Existing research [22] [10] [31] also shows that Prolog can be used to improve the performance of LLMs on logical reasoning tasks. They all integrate LLMs with symbolic solvers to improve logical problem-solving. Their methods first utilizes LLMs to translate a natural language problem into a symbolic formulation, e.g. Prolog. Afterward, a symbolic solver performs inference on the formulated problem. Therefore we chose this approach and try to apply it to the domain of legal texts. More precisely: We choose the German Dentist Fee Schedule (GoZ - Gebührenordnung für Zahnärtze) [1] as a legal text. Our goal is to translate the rules in the GoZ that deal with the correctness of dental invoices into Prolog in an automated way through an LLM and to use the resulting Prolog code in a symbolic solver to decide whether a dental invoice is correct or contains errors according to these very rules of the GoZ.

3. Methodology

3.1. Causal Discovery

To improve the causal discovery capability of LLM, we enhance its capabilities by equipping it with a knowledge base and thus perform retrieval augmentation generation [17]. It turns out that a knowledge base consisting of the paper describing the dataset - which is an ideal knowledge base as the relationships in the data are described in there - improves the capabilities of the LLM, but an web scraped knowledge base, which is more realistic to have, was in our experiments simply not helpful, and even harmed the LLM by providing false or unnecessary information.

3.1.1 Retrieval-augmented Language Models

LLMs still have many limitations like their tendency to hallucinate and generate fictitious responses to user requests. Recent research has shown that this problem can be alleviated by augmenting LLMs with information retrieval (IR) systems (also known as retrieval augmented LLMs) [17]. Applying this strategy, LLMs can generate more factual texts according to the relevant content retrieved by IR systems and by incorporating external knowledge, retrieval-augmented LLMs can answer in-domain questions that cannot be answered by solely relying on the world knowledge stored in parameters [14]. Since the ability of a LLM to do causal discovery depends mainly on the LLM's knowledge and reasoning ability, the idea is that with more knowledge, the LLM should also become better at correctly recognising causal relationships. To test this hypothesis, we equip the LLM with retrieval-based augmentation. This works as follows: We provide the LLM with a knowledge base that contains additional information in textual form. The text in this knowledge base is divided into logically meaningful sections. These sections are converted into embeddings by an LLM and stored in a so-called Vector Store [16]. If the LLM is now queried for the causal relation between two variables, this query is also converted into embeddings. The Vector Store now searches for the text passages whose embeddings have the highest similarity with the embeddings of the query based on the distance between those embeddings. These text passages are then passed to the LLM in the prompt as additional context information [16]. See fig. 2 for an overview of the method.
The reason why this is done is so that the most suitable pieces of text are automatically selected from the entire knowledge base in order to pass them to the LLM with the prompt as context information. Selecting the most suitable pieces of text is important because the entire knowledge base would be too large to be passed to the LLM as input all at once [17]. Now there are different possibilities where the knowledge can be taken from and how exactly the retrieval and processing from the knowledge stored in the Vector Store is done.

Building the Vector store
To provide the LLM with relevant context information, it is essential to use a good knowledge source for the vector store. We have tested two different possibilities.

- Once we tested to what extent the ability for causal discovery improves if we provide the LLM with an optimal knowledge source. As optimal knowledge source we chose the article in which the used dataset is presented, because basically all causal relations should be described here.
- On the other hand, we tested what happens if we let the knowledge source be created automatically using web scraping. However, this is much more complicated than the first variant, as you need a good scraper and good selection of the results. Otherwise you will get inaccurate or false information.

The results are listed under 5.1.

Retrieval and Processing
As mentioned in fig. 2, the query that is given to the LLM is normally used to extract the best matching context information from the vector store using distance based similarity search. This context information is normally put as it is into a prompt and passed to the LLM together with the query. However, this process can be improved:

- Improve Query: When a query is posed, it is questionable whether it is optimally formulated to retrieve

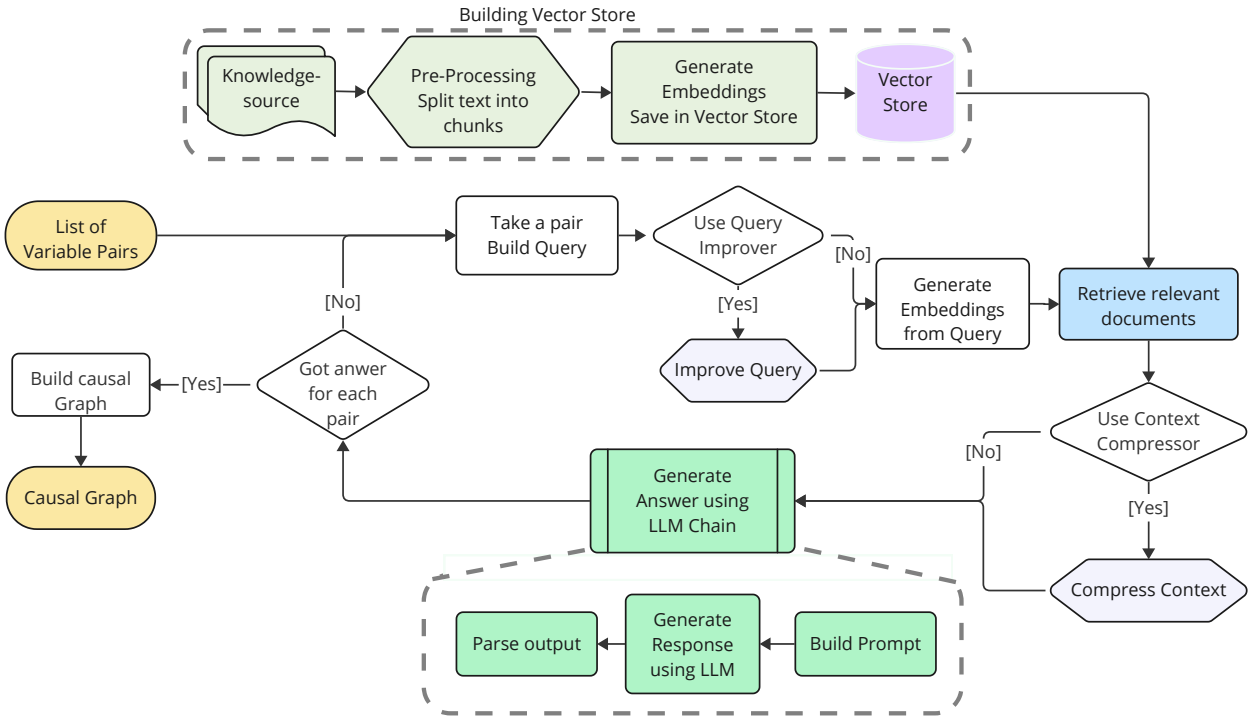

Figure 2. **Overview of the causal discovery LLM pipeline**
First, a vector store is built by loading the knowledge source containing the context information, splitting it into logically related parts, and then converting them into embeddings. If a query is sent to the LLM, this query is also converted into embeddings, and the best matching text pieces are extracted from the vector store using similarity search based on the distance between the query embeddings and text piece embeddings. Optionally, we can insert an extra step in between, where we improve the query to make sure we get all best matching text pieces. The extracted text pieces then form the context information for the prompt. Optionally, the obtained context information can be processed further, for example compressed to reduce the prompt length. The finished prompt, which now contains both the context information and the original query, is then passed to the LLM, which then returns a response. The response contains the causal relationship of the variables as determined by the LLM. This process is repeated for each pair of variables so that a causal graph can be constructed.

exactly the best matching context information from the vector store. Retrieval may produce different results with subtle changes in query wording or if the Embeddings do not capture the semantics of the data well. This problem can be solved with a multi-query retriever. The MultiQueryRetriever automates the process of prompt tuning by using an LLM to generate multiple queries from different perspectives for a given user input query. For each query, it retrieves a set of relevant documents and takes the unique union across all queries to get a larger set of potentially relevant documents. By generating multiple perspectives on the same question, the MultiQueryRetriever might be able to overcome some of the limitations of the distance-based retrieval and get a richer set of results [16]. Another way to solve this problem is by using Forward-Looking Active Retrieval augmented generation (FLARE) [14]. This method actively decides when and what to retrieve using a prediction of the upcoming sentence to anticipate future content. It thereby constructs the appropriate queries by itself based on low-confidence tokens of the predicted sentences.

- Process Context: Instead of passing the context to the LLM as it was retrieved, we can process it even further. A challenge in retrieval is often that the most relevant information for a query may be buried in a text piece with a lot of irrelevant text. Passing the entire piece of text to the LLM can lead to more expensive LLM calls and worse responses. Contextual compression is meant to fix this problem. Instead of immediately returning retrieved text pieces as-is, a Context Compressor can compress them using the context of the given query, so that only the relevant information is returned.

3.2. Legal Text Reasoning

For automatic Reasoning with Legal Texts with the help of LLMs the German GoZ (Gebührenordnung für Zahnärtzte) served as our focal legal text. Our goal was to

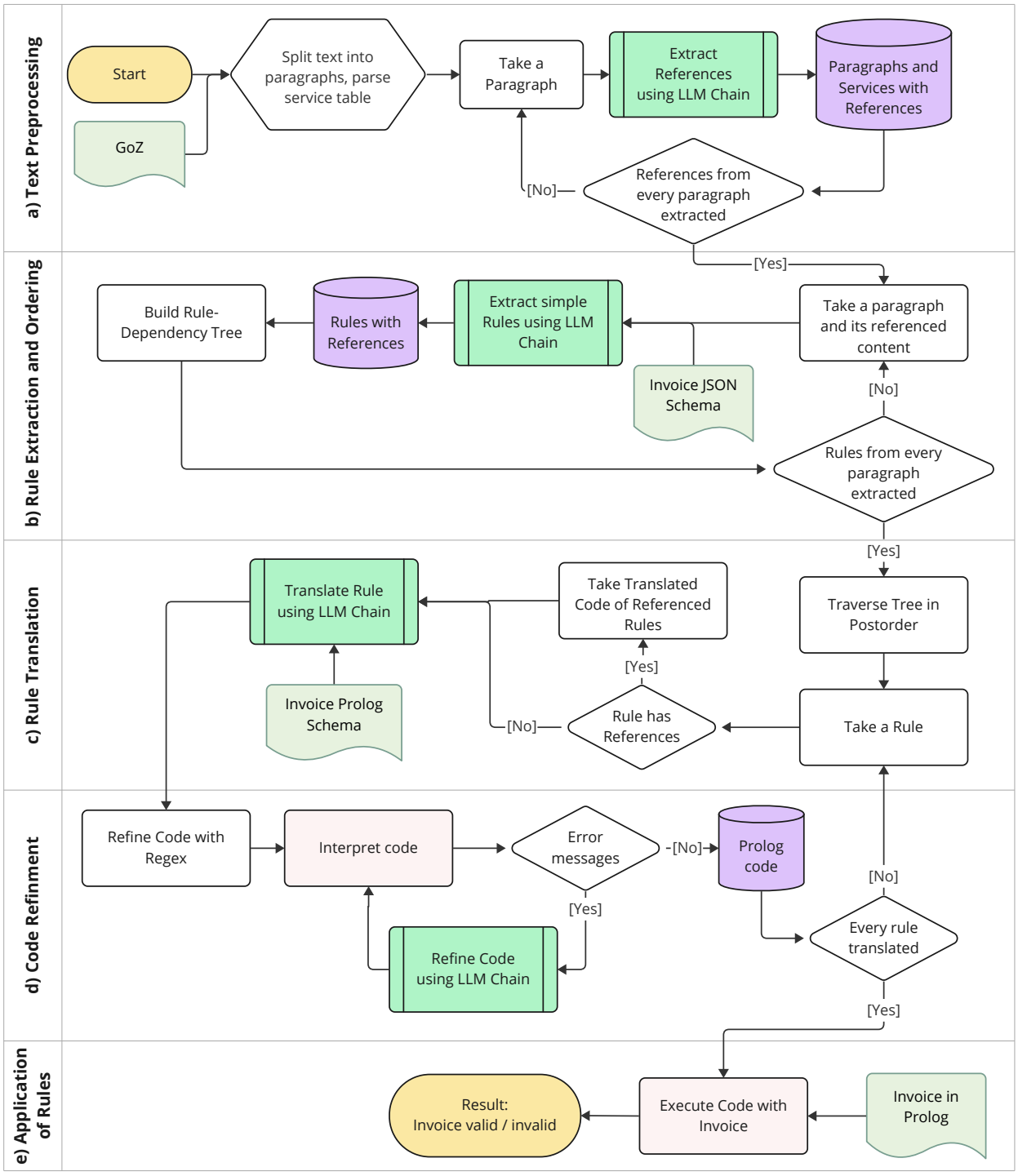

Figure 3. **Overview of the rule extraction LLM pipeline**
(a) First, the legal document is preprocessed into single paragraphs with their references. Also the table of services is parsed.
(b) Individual rules are then extracted from the paragraph and sorted according to the dependency of their references.
(c) The rules are then translated by an LLM. The LLM is asked to carry out the translation using a predefined invoice structure.
(d) Since the translated Prolog code is not always error-free, it is improved using regular expressions and a Prolog interpreter that prompts an LLM with the interpreter errors.
(e) If all rules are translated, they can be applied to an invoice using a solver. The solver returns as a result whether the invoice is invalid according to the translated rules.

convert the rules stated in the GoZ with help of an LLM into some kind of formal representation. It should then be possible to check the correctness of a dentist's invoice using these rules with the help of a solver.
We chose the GoZ because it is a prime example of a real world problem. Insurance policies are becoming more and more complex these days, so that it is hardly possible for people to find out without technical help whether an insurance policy applies to a particular case or not. In the same way, it is not easy to see whether an invoice issued by a dentist is also correct according to the GoZ, or whether the dentist is billing some items incorrectly, and thus charging more than he is entitled to, for example. Checking this manually requires legal expertise and a lot of time. Therefore, the automation of this task is an exciting research direction.
Additionally the GoZ also has a very interesting structure. First comes the general rules, which deal with the invoice as a whole and the general billing of all dental services. These rules are structured as sections, paragraphs and subparagraphs. Subsequently, all dental services are listed in tabular form, according to their GoZ number with their point value (this indicates how much the dentist may charge for them) as well as special additional rules, which usually only apply to these services. In addition, the services are divided into special areas. Each area has its own rules, which usually apply to all services in that area.
We now explain our exact method of converting the legal rules formulated in the GoZ into a formal representation.

Legal texts are often formulated in such a way that translating the legal rules into a formal representation is not a trivial task, neither for human experts and certainly not for LLMs. The reason for this is that legal texts often contain exceptions and substitutions, numerical reasoning, cross-references, and require common sense knowledge [11]. Difficulties here are the correct interpretation and translation of exceptions and substitutions, the correct translation of numerical reasoning in the formal language, correctly putting the text passages from the references into context, and recognising when common sense knowledge is expected and incorporating this correctly. A naive prompting of the LLM to translate a legal text into Prolog does not lead to useful results, which can be seen under 5.2.2. The biggest problems hereby are the inability of LLMs to resolve references correctly, inconsistencies in the code when translating different rules and the tendency of LLMs to hallucinate. Our method solves those problems. An overview of the method can be seen in fig. 3.

a) Text Preprocessing
To simplify the conversion of the legal text to Prolog for the LLM, we carry out a specific Preprocessing-Step on the legal document. We first divide the GoZ into its 3 parts. The first part contains general rules. Only some of them relate directly to the validity of invoices. The second part contains rules that relate only to a specific group of dental services and their billing. The third part consists of a table where each dental service is listed with its number, point value and rules that only apply to this service. For the first and second part, we split the text into individual sections and paragraphs. We then parse the service table into a dataframe. After his, the GoZ is now divided into individual sections and services. We then extract all references to other paragraphs for each paragraph and service using an LLM. As a result, the text is now more structured and it is possible to feed one paragraph after the other to the LLM for further processing, together with all referenced paragraphs.

b) Rule Extraction and Ordering
A common problem with legal texts is that they sometimes contain imprecise, vague rules [28] or rules that cannot be checked programmatically. This is also the case with the GoZ. An example for this is a Rule in §5:

> Within the fee range, the fees shall be determined at reasonable discretion, taking into account the difficulty and time required for the individual service as well as the circumstances of its execution.

The ambiguity here lies in the fact that nowhere is it described exactly how the fees are now determined. The phrase "at reasonable discretion" cannot be translated into programme code based solely on knowledge from the GoZ. Another problem is that the GoZ also contains passages that do not deal with the correctness of dental invoices and are therefore irrelevant for us. We solve this problem by first extracting relevant rules in natural language from the paragraphs and service texts with the help of an LLM. The LLM is given a standardised JSON schema of dental invoices in addition to the text passage from which it has to extract the rules. This schema is required to embed the rules in a standardised context. The LLM is then asked via a special constructed prompt, to extract only specific rules from the text passages.

As mentioned, the GoZ contains three types of rules: rules that relate to the invoice in general R_{general}, rules that relate only to a specific group of services $R_{\text{service-group}}$ and rules that refer to specific dental services R_{service}. Together, they form the set of rules contained in the GoZ: $R_{\text{goz}} = R_{\text{general}} \cup R_{\text{service-group}} \cup R_{\text{service}}$.
For each rule $r \in R_{\text{goz}}$, let $I_{r,\text{correct}}$ be the set of all possible invoices that are valid according to the rule r, and $I_{r,\text{incorrect}}$ be the set of all possible invoices that violate the rule r. Let $I_r = I_{r,\text{correct}} \sqcup I_{r,\text{incorrect}}$ the set of invoices where it is

decidable if Rule r is fulfilled or violated.
However, not every rule $r \in R_{\text{goz}}$ can be applied to an invoice in I_r by a computer program. For a rule to be programmatically applicable to an invoice $i \in I_r$, the following must be fulfilled:
We must be able to verify compliance with or violation of the rule based solely on the information available to us, which means, information contained in the GoZ or schema j of the invoice. If this is not the case, or if, for example, we need additional information that we do not have in order to apply the rule, we cannot work with this rule and therefore do not consider it any further. Since we have a standardised schema j for all invoices, the following must now apply:
To be able to apply a natural language rule $r \in R_{\text{goz}}$ programmatically, we must be able to convert it into a function that can apply the rule to a invoice ($i \in I_r$) and return as a result whether the invoice fulfils or violates the rule $f_{r,j} : I_r \to \{\text{True}, \text{False}\}$. Let $F_{r,j}$ be the set of all such functions $F_{r,j} = \{f_{r,j} \mid f_{r,j} : I_r \to \{\text{True}, \text{False}\}\}$. We now define $P_{r,j}$ as the subset of these functions that represent the rule truthfully:

$$P_{r,j} = \left\{ f_{r,j} \in F_r \;\middle|\; \begin{array}{l} \forall i \in I_{r,\text{correct}} : f_{r,j}(i) = \text{True} \wedge \\ \forall i \in I_{r,\text{incorrect}} : f_{r,j}(i) = \text{False} \end{array} \right\}$$

and $W_{r,j}$ as the subset of functions that represents the rule incorrectly $W_{r,j} = F_{r,j} \setminus P_{r,j}$ We now only want to extract rules where there is at least one correct function for the rule, i.e. where $P_{r,j} \neq \emptyset$
The set of rules that we therefore want to extract is

$$R_{\text{relevant, j}} = \{r \in R_{\text{goz}} \mid P_{r,j} \neq \emptyset\}$$

We now use an LLM with specific prompts (see A.2.2) to extract a set of rules $R_{\text{extracted, j}}$ from the GoZ, which in the optimal case should be equal to $R_{\text{relevant, j}}$:

$$LLM_{\text{extract}} : \text{GoZ} \times \{j\} \to \mathcal{P}(R_{\text{goz}})$$
$$R_{\text{extracted, j}} = LLM_{\text{extract}}(\text{GoZ}, j)$$

$\mathcal{P}(R_{\text{goz}})$ in this case means the power set of R_{goz}. In the following, we will only look at the invoices, whose correctness or invalidity can be determined by those rules: $I = \bigcap_{r \in R_{\text{relevant, j}}} I_r$.
On the one hand, these prerequisites ensure that no rules are translated that would never be applied, on the other hand, they ensure that the translation results in meaningful and executable code and that all Prolog predicates can be initialised using the specified calculation, as uninitialised predicates would lead to runtime errors in Prolog. So those prerequisites filter the pieces of text and also simplifies them, as specific rules have now been extracted from complex legal texts. The extracted rules are now arranged in a dependency tree based on their previously extracted references and traversed in post-order.

c) Rule Translation
The way we constructed $R_{\text{extracted, j}}$, we ensured that for the rule r the function $f_{r,j} \in P_{r,j}$ exists and is computable. As Prolog is Turing Complete [7] we can express $f_{r,j}$ in Prolog code. Each rule $r \in R_{\text{extracted, j}}$ is therefore passed to an LLM in order to translate it into Prolog code:

$$LLM_{translate} : R_{\text{extracted, j}} \times \{j\} \to (I_r \to \{\text{True}, \text{False}\})$$
$$f_{r,j} = LLM_{translate}(r, j)$$

In the following, let $f_{r,j}$ always be the Prolog function of the rule r translated by the LLM. In addition to the textual rule to translate, the LLM also receives the standardised schema j of dental invoices - now in Prolog format - in its prompt. Since we are traversing the reference-based dependency tree in post-order, we can also pass the already translated Prolog code of the referenced paragraphs to the LLM as a reference. We also tell the LLM to define the rules using a certain Prolog schema. The exact Prompt can be seen under A.2.3.
All of this is done to ground the LLM's response. This makes the answers more consistent. The LLM does not invent any new predicates and the structure of the generated code becomes more homogeneous across different rules.
With all rules $r \in R_{\text{extracted, j}}$ now translated into Prolog ($f_{r,j}$) and the way we defined I, we can now automatically check if an invoice $i \in I$ is correct according to the extracted rules ($R_{\text{extracted, j}}$), as for all $i \in I$ it holds that i is correct according to $R_{\text{extracted, j}}$, if and only if for all Rules $r \in R_{\text{extracted, j}}$ the corresponding Prolog function ($f_{r,j}$) returns, that the invoice i is valid ($f_{r,j}(i) = \text{True}$)

$$\begin{aligned} & i \text{ is correct according to } R_{\text{extracted, j}} \\ \iff & \forall r \in R_{\text{extracted, j}} : f_{r,j}(i) = \text{True} \\ \iff & \bigwedge_{r \in R_{\text{extracted, j}}} (f_{r,j}(i)) = \text{True} \end{aligned}$$

To execute $\bigwedge_{r \in R_{\text{extracted, j}}} (f_{r,j}(i))$ in Prolog, however, we must apply a small trick. In Prolog, for a conjunction, all functions must be concatenated with a ",". While for an disjunction , all predicates with the same name are executed sequentially until the first one returns True. The second variant is more practical for us, as we can directly take the individual functions translated by the LLM and do not need to process them further. Therefore, we utilize that $\bigwedge_{r \in R_{\text{extracted, j}}} (f_{r,j}(i)) = \neg \bigvee_{r \in R_{\text{extracted, j}}} (\neg f_{r,j}(i))$ and thus, we can directly adopt each rule $\neg f_{r,j}$ under the predicate *is_invoice_invalid* as translated by the LLM.

d) Code Refinement
As the translated Prolog code is sometimes not error-free, we carry out a refinement step. Frequent errors in the LLM are automatically corrected using regular expressions,

such as converting a "not" into the "\+" character that corresponds to Prolog. The code is then interpreted by a Prolog interpreter. If the interpreter throws errors, the code with the error messages are sent back to an LLM with the request to correct them. This is repeated until the code is error-free, as can be seen in fig. 3.

e) Application of Rules
Now we can apply the rules translated to Prolog to an invoice. The invoices are already available to us as JSON. We can therefore easily translate the JSON structure into a Prolog data object according to the already defined schema, which all rule translations are already based on. Using a Prolog solver such as SWI-Prolog, we can now automatically check for each rule whether the *is_invoice_invalid* predicate is true and whether there is a rule violation.
Under A.2.3 one can see on a view examples what exactly the output is after each step.

4. Implementation

In this chapter, we will discuss the implementation details of our methodology for automatic reasoning with legal texts using Language Learning Models (LLMs). We will focus on the Python-based application that we developed to process and translate the German GoZ (Gebührenordnung für Zahnärzte) document into a formal representation.

a) Text Preprocessing
Using Python, the GoZ PDF gets parsed, and the text gets extracted. Special emphasis is placed on retaining the structure (sections, paragraphs, and subparagraphs). The GoZ is then divided into three distinct parts: general rules, service-specific rules, and a service table. This is done by using regular expressions as this is the simplest way. The textual content is then parsed into a structured pandas DataFrame format, effectively capturing sections, services, and associated rules. After that, an LLM is employed to identify references within each paragraph and service, aiding in understanding interdependencies within the legal text. For working with LLMs, we use the Python framework Langchain, a framework for developing applications powered by language model. It allows us to use more high level interfaces for the interactions with LLMs [16].

b) Rule Extraction and Ordering
LLMs are then used to sift through the legal text, extracting rules, fulfilling our requirements resulting in $R_{extracted}$. These requirements are given the LLM in textual form in its prompt. Rules which are not fulfilling those requirements are discarded. Thereby complex legal language is distilled into simpler rules, easier for both programmatic understanding and LLM processing. Then a dependency tree is constructed based on references, ensuring that rules are processed in a logical order respecting their interdependencies.

c) Rule Translation
Each rule then undergoes a translation into Prolog code via LLMs. The input for this step is the textual rule, and the output is its Prolog counterpart. In order to save tokens when using the OpenAI API, several rules are packed into a prompt separated by specific separator characters. Thus we have a batch-wise processing of the rules. While translating, references to already translated rules are provided as context to the LLM, ensuring consistency and accuracy. The Prolog representation adheres to a predefined schema, aligning with the standard structure of dental invoices.

d) Code Refinement
After that, the initial Prolog code output from LLMs gets refined. Regular expressions and other automated techniques are here employed to correct common syntactical errors. A Prolog interpreter then tests the code, with any errors triggering a revision cycle using LLMs until error-free code is achieved.

e) Application of Rules
Using SWI-Prolog, each rule is then applied to the invoices. The system was designed to flag any violations of the GoZ, indicating invoice invalidity.

4.1. Supplementary Tools and Techniques

Python and Pandas: For data handling and preprocessing.
PyPDF2: To extract text from the PDF version of GoZ.
LangChain: Integrated with OpenAI's GPT models for LLM-based tasks.
Regular Expressions: Used extensively for text manipulation and pattern recognition.
SWI-Prolog: Employed as the Prolog solver for rule application.

5. Results and Discussion

5.1. Causal Discovery

In the following, we evaluate the causal discovery capabilities of gpt-3.5-turbo and gpt-4 with retrieval augmentation. The temperature parameter for those models was always set to zero. We use different knowledge sources and different techniques to improve queries and context described in 3.1.1.

5.1.1 Pairwise Causal Discover on the Tübingen cause-effect pairs dataset

We measure the performance of causal discovery on the Tübingen dataset by counting the pairs whose causal direction was correctly predicted. The results can be viewed in Table 1. It can be seen that both the knowledge base consisting of the dataset paper and an improved prompt setting bring a performance gain. GPT-4 even achieves 100% accuracy with the dataset paper. Only the automatically generated knowledge base performs worse. This is because we only used very simple web scraping to build up the knowledge base and therefore it contained too much incorrect and unnecessary information, which misled the LLM.

5.1.2 Full graph discovery on the arctic sea dataset

For the Arctic-Sea dataset, we measure the performance using various metrics, which have already been explained under . The results can be viewed in table 2. One can also see here that the performance is improved by the knowledge base.

5.2. Legal Text Reasoning

For all our experiments we use GPT-4, because GPT-3.5-turbo was not able to produce usable Prolog code for our purposes. The temperature parameter was always set to zero. To measure how many rules are translated correctly, we applied the following methodology: We extracted the rules relating to the correctness of dental invoices and which are able to be applied programmatically by hand from the legal text resulting in the set of Rules $R_{\text{relevant, j}}$. For each rule $r \in R_{\text{relevant, j}}$ we then prepared a set of sample invoices $S_r \subseteq I_r$ to which the respective rule is applied. Some invoices $i \in S_r$ fulfil the rule ($i \in I_{r,\text{correct}}$) and others violate it ($i \in I_{r,\text{incorrect}}$). The set of sample invoices for a rule is structured in such a way that they can test the rule in all its facets, meaning that for every function $f_{r,j}$ that does not represent a rule r truthfully $f_{r,j} \in W_{r,j}$, there is at least one invoice in this set $i \in S_r$ for which the rule will return the wrong result.

$$\forall f_{r,j} \in W_{r,j} : \exists i \in S_r : \begin{cases} f_{r,j}(i) = \text{True} & \text{if } i \in I_{r,\text{incorrect}} \\ f_{r,j}(i) = \text{False} & \text{if } i \in I_{r,\text{correct}} \end{cases}$$

Therefore we can now test each translated rule $f_{r,j}$ if it recognise all invalid invoices from S_r as invalid and all valid ones as valid. Only if this is the case we consider the rule to be correctly translated ($f_{r,j} \in P_{r,j}$), otherwise we classify the rule as incorrectly translated ($f_{r,j} \in W_{r,j}$). We measure the rule translation accuracy (RT Acc.) for translating the rules correctly as the number of extracted rules that correctly classify all their respective invoices from the sample invoices S_r, divided by the number of extracted rules combined with the relevant rules:

$$\text{RT Acc.} = \frac{\left| \left\{ r \in R_{\text{extracted, j}} \;\middle|\; \begin{array}{l} \forall i \in S_r \cap I_{r,\text{correct}} : f_{r,j}(i) = \text{True} \wedge \\ \forall i \in S_r \cap I_{r,\text{incorrect}} : f_{r,j}(i) = \text{False} \end{array} \right\} \right|}{|R_{\text{relevant, j}} \cup R_{\text{extracted, j}}|}$$

An example of this can be viewed under A.3.5. In our experiments, the LLM also never extracted rules that would not have been relevant, so in the experiments $R_{\text{extracted, j}} \subseteq R_{\text{relevant, j}}$. Due to a lack of resources, we do not do this for all rules, but for a representative subset. In total, we generated 45 different invoices to test a total of 17 different rules. In order to have a baseline for comparison, we use a simple retrieval augmented LLM (RA LLM). The retrieval augmented LLM works in such a way that the LLM is passed one of the example invoices created in the prompt together with the text passage that the example invoice is to test, and the model then only has to determine whether the invoice is valid or not based on the text. This already represents an unrealistically good baseline, as under real conditions one would have to pass all the text of the GoZ to the LLM, as one would not know the correct text passage in advance. Whereas with our method, the code of all rules can easily be appended together and executed as one programme. The results can be seen in table 3. It can be seen here that our method initially performs slightly worse than the baseline. This is not surprising, since if the LLM interprets the rule incorrectly based on the text, it probably also translates the rule incorrectly. In addition, the LLM occasionally makes errors in the prologue syntax and semantics, making translation more difficult than simple reasoning with the rule. However, since the LLM would have to know the correct text passage in advance in order to perform simple inference, this is not always an option in reality, whereas our approach also works with very long legal texts, for example.

It should also be noted that even if the proportion of correctly translated rules appears to be very low, it is not really that bad in reality. As we are only testing a subset of all the rules, we have selected a number of different rules with different levels of difficulty. In the GoZ, however, the proportion of very simple rules is much higher than in our subset. Therefore, the proportion of correctly translated rules would be much higher if all the rules of the GoZ were taken into account.

In addition, we have managed to improve the performance of our method by utilising the findings from the first part of this work. As we were able to show that additional contextual knowledge improves the causal reasoning abilities of LLMs, we utilize this by using an additional document. There is a version of the GoZ which is extended by comments made by the Bundeszahnärztekammer [2]. We now

Prompt Set.	ZS	OS	ZS	OS FLARE	OS + CC	GPT-4 ZS	GPT-4 ZS
Knowledge Base	-	-	DP	DP	Automatic	-	DP
Accuracy	0.83	0.86	0.87	0.94	0.7	0.96	1

Table 1. Causal discovery performance on the Tübingen dataset with different knowledge bases and prompt settings
ZS - Zero Shot, OS - One Shot, CC - Context Compression, DP - Dataset Paper

Prompt Set.	ZS	ZS	ZS	ZS + CC	M-Q-R + CC	FLARE	None (SOTA)
KB	-	DP	DP + Ref. Lit.	DP + Ref. Lit.	DP + Ref. Lit.	DP + Ref. Lit.	-
# Edges	38	24	16	22	16	17	24
NHD	0.35	0.28	0.29	0.29	**0.26**	0.28	0.33
NHD Ratio	0.58	**0.56**	0.66	0.59	0.59	0.62	0.66
Acc.	0.62	0.7	0.68	0.68	**0.71**	0.68	0.63
F1	0.42	**0.44**	0.34	0.4	0.4	0.36	0.33

Table 2. Causal discovery performance on the Arctic-Sea dataset with different knowledge bases and prompt settings. The last columns shows the performance of the best data driven causal discovery algorithm on this dataset.
ZS - Zero Shot, OS - One Shot, CC - Context Compression, M-Q-R - Multi-query-retriever, DP - Dataset Paper, DP + Ref. Lit. - Dataset Paper and Referenced Literature

add these comments to the prompt in order to provide the LLM with more contextual information. This information helps the LLM clearly to translate the rules correctly. What is surprising, however, is that our retrieval augmented LLM baseline does not benefit from the increased context provided through the GoZ commentary. Here, the additional text tends to confuse the LLM, leading to incorrect conclusions. A more detailed analysis can be found at 5.2.1.

5.2.1 Understanding LLM output

On a few examples we will now demonstrate where our method works well and where it produces incorrect translations. Simple to moderately difficult rules are usually translated very well by the LLM. Examples of those can be seen in the Appendix under A.3. Incorrect translations usually occur for the following three reasons. We will always state the reason and give a rule as an example where this applies.

The rule contains terms unknown to the LLM
Rule 0065:

> In addition to the service according to number 0065, conventional impressions according to this fee schedule are not billable for the same half of the jaw or the same anterior tooth area.

The LLM cannot know which dental services are considered conventional impressions and therefore repeatedly translates this regulation incorrectly. However, this is no longer the case if we provide the LLM with the GoZ commentary as additional context. Since the commentary explains which services are considered conventional impressions, the rule is now translated correctly:

```
is_invoice_invalid(Invoice) :-
    Invoice = invoice(Services, _, _, _),
    member(service(0065, _, _, _, _, _, _, Teeth)
           , Services),
    member(tooth(Area, _), Teeth),
    member(service(ConventionalService, _, _, _,
           _, _, _, ConventionalTeeth), Services)
           ,
    member(tooth(Area, _), ConventionalTeeth),
    member(ConventionalService, [5170, 5180,
           5190]),
    print('Invalid invoice: Service 0065 and
           conventional impressions billed for
           the same jaw half or front tooth area
           .'),
    true.
```

The Prolog translation contains semantic errors
Rule 0110:

> The surcharge according to number 0110 can only be charged once per treatment day and only at the single fee rate.

Translation:

```
is_invoice_invalid(Invoice) :-
    Invoice = invoice(Services, _, _, _),
    findall(Date, member(service(0110, Date,
           Multiplier, _, _, _, _, _), Services),
            Dates),
    sort(Dates, SortedDates),
    length(Dates, Length),
    length(SortedDates, SortedLength),
    (Length > SortedLength ; Multiplier \= 1),
    print('The surcharge for service 0110 can
           only be charged once per treatment day
            and only at the simple rate.'),
    true.
```

Here the LLM makes a beginner's error in Prolog and does not bind the multiplier variable in line 8. Therefore, an in-

	R.A. LLM	Our Method	Our Method + Commentary	R.A. LLM + Commentary
correct classified invoices	35.8 ± 1.8	31.6 ± 0.5	37.2 ± 0.8	30.0 ± 2.1
wrong classified invoices	9.2 ± 1.8	13.4 ± 0.5	7.8 ± 0.7	15.0 ± 2.1
correct classified rules	10.2 ± 1.5	7.2 ± 0.4	11.4 ± 0.8	5.8 ± 1.2
wrong classified rules	10.2 ± 1.5	9.8 ± 0.4	5.6 ± 0.8	11.2 ± 1.17
RT Acc.	0.6 ± 0.09	0.42 ± 0.02	0.67 ± 0.05	0.34 ± 0.07

Table 3. Accuracy in recognising the validity or invalidity of the manually constructed sample invoices and the resulting accuracy in interpreting rules. For each of the methods we perform five different runs and report mean and standard deviation of those runs. Our Method performs slightly worse than the retrieval augmented LLM baseline. However when we add the GoZ commentary as additional context, our method beats the baseline.

voice that has the service 0110 with a multiplier other than 1 is not classified as invalid. The rule would be translated correctly if the multiplier variable in the `findall` Predicate was bound correctly like this:

```
is_invoice_invalid(Invoice) :-
    Invoice = invoice(Services, _, _, _),
    findall(Date-Multiplier, member(service(0110,
            Date, Multiplier, _, _, _, _),
           Services), DateMultiplierList),
    sort(DateMultiplierList,
           SortedDateMultiplierList),
    length(DateMultiplierList, Length),
    length(SortedDateMultiplierList, SortedLength
           ),
    (Length > SortedLength ;
           not_all_multipliers_one(
           SortedDateMultiplierList)),
    print('The surcharge for service 0110 can
           only be charged once per treatment day
            and only at the simple rate.'),
    true.

not_all_multipliers_one([]).
not_all_multipliers_one([_-Multiplier|Rest]) :-
    (Multiplier =\= 1 ; not_all_multipliers_one(
           Rest)).
```

The LLM does not manage to translate the whole meaning of the rule

Rule 2030:

> The service according to number 2030 can be billed no more than once per session for one half of the jaw or one anterior tooth area for special preparation measures and no more than once for special measures when filling cavities.

Translation:

```
is_invoice_invalid(Invoice) :-
    Invoice = invoice(Services, _, _, _),
    findall(Session, (member(service(2030,
           Session, _, _, _, _, _, _), Services))
           , Sessions),
    maplist(count_occurrences(Services), Sessions
           , Counts),
    max_list(Counts, Max),
    Max > 2,
    print('Service 2030 charged more than twice
           in a single session.'),
    true.

count_occurrences(Services, Session, Count) :-
    findall(_, (member(service(2030, Session, _,
           _, _, _, _, _), Services)),
           Occurrences),
    length(Occurrences, Count).
```

With this rule, for example, the LLM does not manage to cover the fact that the service may not be billed more than twice only per jaw half, although the jaw half to which a service was applied is included in the billing and the LLM was also given this information in the prompt (see A.2.3). The LLM translates the rule as if the service may not be billed more than twice at all, which is not the case.

So overall, we can classify the errors that the LLM makes into two classes, on the one hand wrong translations, where the prologue code contains errors, but it is obvious that the LLM has understood the rule correctly, or on the other hand wrong translations, where the LLM has misunderstood the rule and therefore translated it incorrectly. The distribution of errors can be seen in the table 4.

Baseline with GoZ commentary

The retrieval augmented LLM baseline does not benefit from the additional context information given through the GoZ commentary as shown in 3. The main reason for this is that the additional text confuses the LLM, leading to incorrect conclusions, an example of which is Rule 0120:

> The surcharge according to number 0120 amounts to 100% of the basic fee rate for the respective service, but not more than 68 euros. The surcharge according to number 0120 can only be calculated once per treatment day.

An invoice was submitted to the LLM in which service 0120 was billed on two different days. The invoice is therefore valid, but the reasoning of the LLM is wrong, therefore the LLM classifies this invoice wrongly as invalid:

> Reasoning: The rule states that the surcharge for the use of a laser (GOZ No. 0120) is only chargeable once per treatment day. In the given invoice,

	rule misunderstood	Prolog code error
Our Method	65 %	35 %
Our Method + Commentary	73 %	27 %

Table 4. Distribution of the reason why a rule was translated incorrectly.

> the service with GOZ No. 0120 is charged twice on the same day (2012-02-12 and 2012-03-12). Therefore, the invoice does not comply with the rule. Answer: No

5.2.2 Ablation Studies

In the following, we conduct a few ablation studies to justify the exact choice of our method.

Rule Extraction
Without prior rule extraction, our method performs much worse, as can be seen under table 5. This is mainly due to the fact that the LLM has difficulty translating the complex legal text into simple Prolog rules. Therefore, the extra rule extraction step is necessary beforehand.

References
Especially with rule extraction, it leads to an improved quality of the extracted rules when we include the referenced paragraphs as additional context. An example of this is § 10:
Rule extraction without references:

> If a service is charged in accordance with § 6 (1), the correspondingly valued service must be described in a way that is comprehensible to the payer and labelled "corresponding" as well as the number and name of the service deemed to be equivalent.

Rule extraction with references:

> If a service is charged according to § 6 paragraph 1, the service must be described in a way that is understandable for the person liable to pay. This can be checked by verifying that the "Description" field for each service is not empty.
>
> The service must be marked with the note "corresponding" as well as the number and the designation of the service considered equivalent. This can be checked by verifying that the "ServiceNumber" field for each service is not empty and that the "Description" field contains the word "corresponding".

It can be clearly seen that the extracted rules where the references where given are better broken down and easier to understand.
The arrangement of the individual rules in a dependency tree and the inclusion of translation of the referenced rules in the prompt when translating the new rule unfortunately has, in our experience, a negligible influence on the quality of the translation. However, we nevertheless leave this step in our method, as it can still contribute to making the translations more homogeneous across different rules.

Grounding on the Invoice Structure The grounding of the translations of the rules to a fixed invoice structure specified in the prompt makes the rules more homogeneous and produces less wrong translations. Let's look at rules from §5 as an example:

> The amount of the individual fee is based on one to three and a half times (multiplier) the fee rate. The fee rate is the amount that results when the number of points for the individual service in the fee schedule is multiplied by the point value. The point value is 5.62421 cents.
> The 2.3-fold fee rate reflects the average service in terms of difficulty and time required; exceeding this fee rate is only permitted if this is justified by special features of the assessment criteria mentioned in sentence 1; services with a below-average degree of difficulty or time required are to be charged at a lower fee rate.

Translation with grounding on invoice structure:

```
calculate_charge(PointScore, Multiplier, Charge)
        :-
    PointValue = 0.0562421,
    CalculatedCharge is round(PointValue *
           Multiplier * PointScore * 100) / 100,
    Charge = CalculatedCharge.
is_invoice_invalid(Invoice) :-
    Invoice = invoice(Services, _, _, _),
    member(service(_, _, Multiplier, PointScore,
           Charge, _, _, _), Services),
    \+ calculate_charge(PointScore, Multiplier,
           Charge),
    print('Invalid invoice: The charge for a
           service does not match the calculated
           charge.'),
    true.

is_multiplier_valid(Multiplier, Justification) :-
    Multiplier =< 2.3;
    (Multiplier > 2.3, Justification \= '').
is_invoice_invalid(Invoice) :-
    Invoice = invoice(Services, _, _, _),
```

	Our Method	Our Method without Rule Extraction
correct classified invoices	31.6 ± 0.5	28 ± 1.2
wrong classified invoices	13.4 ± 0.5	17 ± 1.2
correct classified rules	7.2 ± 0.4	6 ± 0.9
wrong classified rules	9.8 ± 0.4	11 ± 0.9
accuracy	0.42 ± 0.02	0.35 ± 0.04

Table 5. Accuracy in recognising the validity or invalidity of the manually constructed sample invoices and the resulting accuracy in interpreting rules. For each of the methods we perform five different runs and report mean and standard deviation of those runs. When we omit the rule extraction step from our Method, it can be seen that the results performance decreases.

```
    member(service(_, _, Multiplier, _, _, _,
           Justification, _), Services),
    \+ is_multiplier_valid(Multiplier,
           Justification),
    print('Invalid invoice: The multiplier for a
           service exceeds 2.3 without
           justification.'),
    true.
```

Translation without grounding:

```
point_value(0.0562421).
% Predicate to calculate the fee for a service
fee(Service, Multiplier, Fee) :-
    point_value(PointValue),
    service_points(Service, Points),
    Fee is round((Points * PointValue *
           Multiplier)*100)/100,
    Fee >= round((Points * PointValue)*100)/100,
    Fee =< round((Points * PointValue * 3.5)*100)
           /100.
% Predicate to check if an invoice is invalid
       based on Rule 1
is_invoice_invalid(Invoice) :-
    member(Service-Multiplier, Invoice),
    \+ fee(Service, Multiplier, _),
    print('Invoice is invalid: Fee for service is
            not within allowed range.').

This rule states that the 2.3-fold fee represents
        the average service. If the fee exceeds
       this, it must be justified by the
       difficulty or time required for the
       service. Services with below-average
       difficulty or time requirement should be
       calculated with a lower fee. This rule is
       subjective and depends on the individual
       service, so it cannot be translated into
       Prolog without making assumptions.
```

One can see that without grounding, the translation does not adhere to a structure, so that translations of different rules end up not being compatible with each other. In addition, some rules are not translated because the LLM is not sure what is really given in the calculation. The translations without grounding are therefore not usable.

Our approach can also be applied to other kind of contracts or invoice structures. For example, we use rule 1000 to show that the translation of the rules automatically adapts to the modified invoice structure.

> The service according to number 1000 is billable only once within a year.

Translation with our invoice structure:

```
is_invoice_invalid(Invoice) :-
    Invoice = invoice(Services, _, _, _),
    findall(Service, (member(service(Service,
           date(Year, _, _), _, _, _, _, _, _),
           Services), Service = 1000),
           Services1000),
    length(Services1000, Count1000),
    Count1000 > 1,
    print('Service 1000 is charged more than once
            in a year.'),
    true.
```

Now we adjust the invoice structure by adding a history of all services billed in past invoices. The translation then automatically adjusts as follows:

```
is_invoice_invalid(Invoice) :-
    Invoice = invoice(Services, _,
           Service_History, _, _),
    findall(S, (member(service(S, date(Year, _, _
           ), _, _, _, _, _, _), Services), S =
           1000), Current_Services_1000),
    findall(S, (member(service(S, date(Year, _, _
           ), _, _, _, _, _, _), Service_History)
           , S = 1000), History_Services_1000),
    length(Current_Services_1000,
           Count_Current_1000),
    length(History_Services_1000,
           Count_History_1000),
    Total_1000 is Count_Current_1000 +
           Count_History_1000,
    Total_1000 > 1,
    print('The service with number 1000 is
           charged more than once in a year.'),
    true.
```

Code Refinement Parts of the code refinement step are essential to be able to execute the code correctly. For example, the refinement step with regular expressions is very important to ensure that known Prolog errors are immediately eliminated. Although the refinement LLM chain is only used for approx. 2% of the rules, it ensures that the code is always executable.

5.2.3 Generalisability of our method

Even though we have developed and tested our method primarily for use on the GoZ, our approach can be easily generalised to other domains. This means that other legal texts that deal with other types of contracts, can also be parsed to determine whether a contract comply with the given rules. To do this, only the preprocessing has to be adapted to the respective text, and the prompts have to be slightly adapted so that they no longer contain the schema of a dentist's invoice, but that of the respective contract. After these adjustments, our method can be applied to the new text.

5.2.4 Limitations

While the presented research demonstrates significant advancements in utilizing Large Language Models (LLMs) for causal discovery and legal text interpretation, some limitations remain. Understanding these limitations is crucial for guiding future research and practical applications.

Model Dependence and Generalization
The performance of our methods heavily relies on the capabilities of the LLMs used, specifically GPT-4. Different LLMs or versions might yield significantly different results, particularly in terms of accuracy in causal discovery and legal text translation.

Quality of Knowledge Base for Causal Discovery
For the causal discovery task, the quality of the knowledge base used for retrieval-augmented generation plays a pivotal role. Optimal results were achieved using a knowledge base consisting of the scientific paper describing the dataset. However, creating a similarly effective knowledge base through automated means, such as web scraping, proved challenging. The quality of automatically generated knowledge bases is variable and often includes irrelevant or inaccurate information, which can mislead the LLM and degrade performance.

Translation Accuracy in Legal Text Interpretation
Even though an LLM can obviously be very helpful in converting legal text into a formal language, it is not error-free, as shown in 5.2.1. It can therefore serve very well as an assistant and enable people without legal training to convert legal rules into a formal representation, but a human should always check the result.

Computational Resources and Running Time
The methodology used, particularly for translating an entire legal text into a logical language, requires numerous calls to the LLM. This process, involving both the creation of prompts and generation, consumes a substantial number of tokens, which can lead to significant costs when using the OpenAI API, depending on the price per token. Additionally, the runtime varies depending on the used model, meaning that translating the entire text with certain models could result in considerable processing time.

6. Future Work

Since, as mentioned, there exists still some limitations when it comes to translating a legal text into a formal language, a number of future research directions are possible. For example, one could investigate whether erroneous rule translations can be reduced even further by having another LLM generate test cases, to which the translated rules are then applied. In this way, an initial sanity check could be carried out, the result of which could also be taken into account in the code refinement step. One could also try to mitigate the inherent disadvantages of semantic parsing of the rules into a formal language, such as the fact that a tiny error in the code immediately leads to an incorrect result, by not only relying on the symbolic solver in the final reasoning, but also including LLMs here. Further experiments would be exciting to find out to what extent open source models can be used with our method. Also, as LLMs continue to evolve, our method becomes more applicable as the semantic parsing capabilities of LLMs improve, reducing the error rate of incorrect translations. Through reducing the errors that occur in semantic parsing and though improved capabilities of LLMs in the future, it may be possible to develop this method into a fully automated contract review and due diligence application. This methodology could also be interesting for other domains, for example it could be applied to technical manuals or also to the field of medical documentation and diagnostics, by translating complex medical texts into more structured and formal languages.

7. Conclusion

We have shown that the ability of LLMs to perform causal discovery can be improved by retrieval augmented generation with a meaningful knowledge base. In particular, a relatively simple knowledge base, consisting only of the scientific paper describing the dataset, has brought an improvement. However, since such an optimal knowledge base is usually not available in reality, we have also experimented with having the knowledge base created automatically. There is still room for improvement here, especially when scraping, selecting and filtering the information. Overall, however, we have shown that LLMs can be used as an additional tool for causal discovery, preferably alongside numerical data-based systems and as assistants.
Furthermore, we have shown a method to perform legal reasoning with the help of LLMs. More precisely, we use LLMs to translate a legal text into the logical programming language Prolog and can use the translation to conclude

whether a contract whose formalities have been described in the legal text is correct according to the rules in this text. We demonstrate this using the German GoZ (Gebührenordnung für Zahnärtze) and dental invoices as a case study and show that even if the translations of the LLM are not always perfect, this method can at least be used as a very useful assistant to convert a legal text into a formal representation.

References

[1] Gebührenordnung für zahnärzte (goz). Bundeszahnärztekammer, 2020. Letzte Überarbeitung dieser Seite 06.10.2020. 4

[2] Gebührenordnung für zahnärzte (goz) kommentar der bundeszahnärztekammer (bzÄk). Bundeszahnärztekammer, 2020. Letzte Überarbeitung dieser Seite 06.10.2020. 10

[3] Jessica D Blankshain and Andrew L Stigler. Applying method to madness: A user's guide to causal inference in policy analysis (summer 2020). *Texas National Security Review*, 2020. 2

[4] Amina Boufrida and Zizette Boufaida. Rule extraction from scientific texts: Evaluation in the specialty of gynecology. *Journal of King Saud University-Computer and Information Sciences*, 34(4):1150–1160, 2022. 3

[5] Ivan Bratko. *Prolog programming for artificial intelligence*. Pearson education, 2001. 1, 3

[6] Samuel R Buss. An introduction to proof theory. *Handbook of proof theory*, 137:1–78, 1998. 3

[7] Danny Crookes. Using prolog to present abstract machines. *ACM SIGCSE Bulletin*, 20(3):8–12, 1988. 8

[8] Claire Donnat and Susan Holmes. Tracking network dynamics: A survey using graph distances. *The Annals of Applied Statistics*, 12(2):971–1012, 2018. 2

[9] Thomas A Glass, Steven N Goodman, Miguel A Hernán, and Jonathan M Samet. Causal inference in public health. *Annual review of public health*, 34:61–75, 2013. 2

[10] Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Luke Benson, Lucy Sun, Ekaterina Zubova, Yujie Qiao, Matthew Burtell, David Peng, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, Ansong Ni, Linyong Nan, Jungo Kasai, Tao Yu, Rui Zhang, Shafiq R. Joty, Alexander R. Fabbri, Wojciech Kryscinski, Xi Victoria Lin, Caiming Xiong, and Dragomir Radev. Folio: natural language reasoning with first-order logic. *CoRR*, abs/2209.00840, 2022. 1, 4

[11] Nils Holzenberger, Andrew Blair-Stanek, and Benjamin Van Durme. A dataset for statutory reasoning in tax law entailment and question answering. In *Proceedings of the 2020 Natural Legal Language Processing (NLLP) Workshop*, San Diego, US, 2020. 1, 3, 7

[12] Nils Holzenberger and Benjamin Van Durme. Connecting symbolic statutory reasoning with legal information extraction. In *Proceedings of the Natural Legal Language Processing Workshop 2023*, pages 113–131, 2023. 1, 3

[13] Yiyi Huang, Matthäus Kleindessner, Alexey Munishkin, Debvrat Varshney, Pei Guo, and Jianwu Wang. Benchmarking of data-driven causality discovery approaches in the interactions of arctic sea ice and atmosphere. *Frontiers in big Data*, 4:642182, 2021. 2

[14] Zhengbao Jiang, Frank Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992, Singapore, Dec. 2023. Association for Computational Linguistics. 4, 5

[15] Emre Kiciman, Robert Osazuwa Ness, Amit Sharma, and Chenhao Tan. Causal reasoning and large language models: Opening a new frontier for causality, April 2023. 1, 2

[16] Langchain. LangChain: A framework for production-ready nlp. https://github.com/langchain-ai/langchain, 2022. 4, 5, 9

[17] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc., 2020. 4

[18] Stephanie Long, Tibor Schuster, and Alexandre Piché. Can large language models build causal graphs? In *NeurIPS 2022 Workshop on Causality for Real-world Impact*, 2022. 2

[19] Joris M Mooij, Jonas Peters, Dominik Janzing, Jakob Zscheischler, and Bernhard Schölkopf. Distinguishing cause from effect using observational data: methods and benchmarks. *The Journal of Machine Learning Research*, 17(1):1103–1204, 2016. 2

[20] Ha-Thanh Nguyen, Minh-Phuong Nguyen, Thi-Hai-Yen Vuong, Minh-Quan Bui, Minh-Chau Nguyen, Tran-Binh Dang, Vu Tran, Le-Minh Nguyen, and Ken Satoh. Transformer-based approaches for legal text processing: Jnlp team-coliee 2021. *The Review of Socionetwork Strategies*, 16(1):135–155, 2022. 3

[21] Ha-Thanh Nguyen, Fungwacharakorn Wachara, Fumihito Nishino, and Ken Satoh. A multi-step approach in translating natural language into logical formula. In *Legal Knowledge and Information Systems*, pages 103–112. IOS Press, 2022. 3

[22] Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3806–3824, Singapore, Dec. 2023. Association for Computational Linguistics. 1, 4

[23] Nathaniel Popper. Causal inference: A guide for policymakers. https://simons.berkeley.edu/news/causal-inference-guide-policymakers, 2022. Accessed: 2023-12-06. 2

[24] Juliano Rabelo, Mi-Young Kim, and Randy Goebel. Combining similarity and transformer methods for case law entailment. In *Proceedings of the seventeenth international*

conference on artificial intelligence and law, pages 290–296, 2019. 3

[25] Ken Satoh. Proleg: Practical legal reasoning system. In *Prolog: The Next 50 Years*, pages 277–283. Springer, 2023. 1, 3

[26] Ken Satoh, Kento Asai, Takamune Kogawa, Masahiro Kubota, Megumi Nakamura, Yoshiaki Nishigai, Kei Shirakawa, and Chiaki Takano. Proleg: an implementation of the presupposed ultimate fact theory of japanese civil code by prolog technology. In *JSAI international symposium on artificial intelligence*, pages 153–164. Springer, 2010. 1, 3

[27] Ken Satoh, Takamune Kogawa, Nao Okada, Kentaro Omori, Shunsuke Omura, and Kazuki Tsuchiya. On generality of proleg knowledge representation. In *Proceedings of the 6th international workshop on juris-informatics (JURISIN 2012), Miyazaki, Japan*, pages 115–128, 2012. 1, 3

[28] Marek J. Sergot, Fariba Sadri, Robert A. Kowalski, Frank Kriwaczek, Peter Hammond, and H Terese Cory. The british nationality act as a logic program. *Communications of the ACM*, 29(5):370–386, 1986. 1, 3, 7

[29] David M Sherman. A prolog model of the income tax act of canada. In *Proceedings of the 1st international conference on Artificial intelligence and law*, pages 127–136, 1987. 1, 3

[30] Pengzhou Wu and Kenji Fukumizu. Causal mosaic: Cause-effect inference via nonlinear ica and ensemble method. In *International Conference on Artificial Intelligence and Statistics*, pages 1157–1167. PMLR, 2020. 2

[31] Yuan Yang, Siheng Xiong, Ali Payani, Ehsan Shareghi, and Faramarz Fekri. Harnessing the power of large language models for natural language to first-order logic translation. In *Submitted to The Twelfth International Conference on Learning Representations*, 2023. under review. 1, 4

[32] Matej Zečević, Moritz Willig, Devendra Singh Dhami, and Kristian Kersting. Causal parrots: Large language models may talk causality but are not causal. *Transactions on Machine Learning Research*, 2023. 1

[33] May Myo Zin, Ha Thanh Nguyen, Ken Satoh, Saku Sugawara, and Fumihito Nishino. Improving translation of case descriptions into logical fact formulas using legalcasener. In *Proceedings of the Nineteenth International Conference on Artificial Intelligence and Law*, pages 462–466, 2023. 3

Appendices

A. Causal Discovery

A.1. Prompts

Tübingen dataset:

```
Which cause-and-effect relationship is more likely?
A. changing {X} causes a change in {Y}.
B. changing {Y} causes a change in {X}.
 L e t s  work this out in a step by step way to be sure that we have the right answer. Then provide your
        final answer within the tags <Answer>A/B</Answer>.
```

Arctic-Sea dataset:

```
Given the following question and context, answer the question thinking step by step. If the question is
        not answered with the context, use your own knowledge to answer the question. Provide your
       final answer at the end within the tags <Answer>yes/no</Answer>.

Remember, solve the question by thinking step by step.

> Question: Is {X} a direct cause of {Y}?
> Context:
>>>
{context}
>>>
```

A.2. Prompts

A.2.1 Extract References

```
Given following text which is a part of the German GOZ:
```
{text}
```

Extract references to other paragraphs if mentioned in the text.

Write the result as a list of references like for example:
References: 5 (1), 4 (2)
```

A.2.2 Extract Rules

```
Given following text which is a part of the German GOZ:
```
{{ text }}
```
{{ references }}

Extract from this text only rules that both state conditions or restrictions that should be adhered to
       when creating an invoice and can be checked automatically by some code, corresponding to an
       invoice with the following schema:
{invoice_json_schema}

Extract only rules from the text that can be applied to a invoice with this structure.
If you do not find such rules, just return <NO RULES>.
Return the rule in text form.
Separate the rules with ---.
e.g.:
Rules:
---
Rule 1
```

```
---
Rule 2
---
...


Rules:
```

A.2.3 Translate Rules

System prompt:

```
Your are an expert in the logical programming language Prolog and of the german GoZ. The user will give
        you tasks where you have to write Prolog code related to the GoZ.
Please wrap the code in the following way:
```prolog
<your code>
```
```

Basic user prompt:

```
Given this rules from the german GOZ:
---
{text}

And this predicates:
```prolog
invoice(
    % A Service is defined trough its Service Number (Service), the date / session when it was rendered
           , its multiplier, the teeth it is applied to, ...
    [service(Service, date(Year, Month, Day), Multiplier, PointScore, Charge, Description,
           Justification, [tooth(Area, Postion), ...]), ... ],
    % Billed Material Costs are defined trough the Section of the GOZ which allows its billing, the
           date / session when it was rendered, its multiplier, ...
    [material_cost(GOZ_Section, date(Year, Month, Day), Multiplier, Count, Charge, Description,
           Justification, [tooth(Area, Postion), ...]), ... ],
    invoice_date(date(Year, Month, Day)),
    invoice_amount(InvoiceAmount)
).
% Charge = (round(PointValue in Euros * Multiplier * PointScore * 100)) / 100
% A Tooth on which a Service is applied is uniquely defined by the combination of its area and position
        in this area: tooth(Area, Postion) nummerated after the FDI-Zahnschema
% Area = 1 -> Right upper jaw
% Area = 2 -> Left upper jaw
% Area = 3 -> Left lower jaw
% Area = 4 -> Right lower jaw
% Position is exact position of this tooth in its area (from 1 to 8)
% Example invoice
invoice(
    [
        service(11111, date(2023, 10, 01), 1.0, 100, 5.62, 'Teeth Cleaning', '', [tooth(1,7)]),
        service(11111, date(2023, 10, 01), 1.0, 100, 5.62, 'Teeth Cleaning', '', [tooth(1,6)]),
        service(987654, date(2023, 10, 01), 3.5, 200, 39.37, 'Tooth Filling', 'Very time consuming', [
               tooth(1,7), tooth(1,6)]),
        service(123456, date(2023, 10, 02), 2.3, 150, 19.40, 'Consultation', '', [])
    ],
    [
        material_cost('   4 (1)', date(2023, 10, 02), 1.0, 1, 40.00, 'Fill Material', 'Auslagen', [
               tooth(1,7), tooth(1,6)])
    ]
    invoice_date(date(2023, 10, 04),
    invoice_amount(110.01)
).
```
The {rule_count} rules indicate conditions and restrictions stated in the GOZ that should be adhered to
        when creating an invoice.
```

````
We want to mark invoices as invalid if they do not adhere to those rules.
Which of those rules can you translate into prolog code given following restriction:
Only translate the rules as they are, do not include any examples.
Be as accurate as possible.
You must be able to exactly define every new predicate based on the given ones.
Do not make any assumptions which are not given.
Translate the rules where a translation given the restrictions is possible.
If you can not translate the ful rule, translate as much as possible, but do not invent something.
       Maybe you can translate a part of the rule.
Keep the code as simple as possible.
To determine if an invoice is correct, use the predicate
```prolog
is_invoice_invalid(Invoice) :-
    Invoice = invoice(Services, Material_Costs, invoice_date(date(Year, Month, Day), invoice_amount(
           InvoiceAmount)),
    % The corresponding rule (only one rule for each predicate)
    % Please double check that the prolog code you write in here is semantically equivalent to the rule
            in the text.
    % Keep the code as simple as possible.
    print(...), % When rule is not fulfilled and therefore the invoice is invalid, print the reason for
            the invalidity of the invoice
    true. % Then the invoice is invalid.
```
Use this predicate to check each rule individually. Before you translate a rule, think about it.
Expected format:
---
Rule 1:
Thoughts: ...
Translation:
```prolog
% code here (is_invoice_invalid/1 predicate with potential helper predicates for this rule)
```
---
Rule 2:
Thoughts: ...
Translation:
```prolog
% code here (is_invoice_invalid/1 predicate with potential helper predicates for this rule)
```
---
...

When you can not translate leaf the code section empty like this:
---
Rule X:
Thoughts: ...
Translation:
```prolog
```
---

So you should have a block for each of the {rule_count} given rules at the end.
For translated rules include print/1 statements, so that the user can see the reason for the invalidity
        of the invoice.
When you calculate with currencies, always round the result to two digits with round(X*100)/100 and
       always calculate in Euros (even after summation).
I only want to have exactly defined rules.
First, think step by step how you would translate a rule, then translate it. Explain your thoughts step
        by step.
````

When the translated rules do reference other rules, the following is appended on the basic user prompt:

```
The rules references some already translated section.
As references for you, here are the translations of the referenced sections:
{references}
```

When we also use the GoZ commentary as additional context information, the following is also appended:

```
In the following I will give you some more context text. This text is only intended to help you
       understand the context in which you are translating the rule.
Context:
>>>
{commentary}
>>>
```

A.2.4 Refinement of Translated Code

```
Your prolog code
'''prolog
{text}
'''
has produced some Errors. Please correct the Errors.
Errors:
{error_msg}
```

A.3. Examples

A.3.1 GoZ Text

Service rules 0120:

> Surcharge for the use of a laser laser for the services according to 2410, 3070, 3080, 3210, 3240, 4080, 4090, 4100, 4130, 4133 and 9160. The surcharge according to number 0120 amounts to 100 per cent of the simple fee rate for the service in question, but not more than EUR 68. The surcharge according to number 0120 can only be charged once per treatment day.

Service rules 2220:

> Restoration of a tooth with a partial crown with retention grooves or boxes or with pinledges including reconstruction of the restoration of the entire occlusal surface, also restoration of a tooth with a veneer. In addition to the services according to numbers 2200 to 2220, the services according to numbers 2050 to 2130 are not billable. The service according to number 2210 is not billable in connection with implants. The following dental services are covered by the services according to numbers 2150 to 2170 and 2200 to 2220: Preparation of the tooth or implant, determination of the relationship, impressions, try-ins, provisional insertion, permanent insertion of the inlay filling or crown or partial crown or veneer, follow-up check and corrections. The service according to number 2200 also includes screwing and covering with filling material. Crowns according to numbers 2200 to 2220 include crowns (full and partial crowns) of any dental design.

A.3.2 Preprocessed Text

A.3.3 Extracted Rules

Rules 0120:

> Surcharge for the use of a laser for the services according to numbers 2410, 3070, 3080, 3210, 3240, 4080, 4090, 4100, 4130, 4133 and 9160.
>
> The surcharge according to number 0120 is 100 per cent of the simple fee rate for the service in question, but not more than EUR 68.
>
> The surcharge according to number 0120 can only be charged once per treatment day.

Rules 2220:

In addition to the services according to numbers 2200 to 2220, the services according to numbers 2050 to 2130 are not billable.

The service according to number 2210 is not billable in connection with implants.

The following dental services are covered by the services according to numbers 2150 to 2170 and 2200 to 2220: Preparation of the tooth or implant, determination of the relationship, impressions, try-ins, provisional insertion, permanent insertion of the inlay filling or crown or partial crown or veneer, follow-up check and corrections.

The service according to number 2200 also includes screwing and covering with filling material. Crowns according to numbers 2200 to 2220 include crowns (full and partial crowns) of any dental design. If the services end with the preparation of a tooth or the taking of an impression for an implant, half of the respective fee is chargeable.

A.3.4 Translated rules

0120:

```
is_invoice_invalid(Invoice) :-
    Invoice = invoice(Services, _, _, _),
    member(service(0120, _, _, _, Charge, _, _, _), Services),
    Charge > 68,
    print('The surcharge for service 0120 cannot be more than 68 Euros.'),
    true.
is_invoice_invalid(Invoice) :-
    Invoice = invoice(Services, _, _, _),
    findall(Date, member(service(0120, Date, _, _, _, _, _, _), Services), Dates),
    sort(Dates, SortedDates),
    length(Dates, Length),
    length(SortedDates, SortedLength),
    Length > SortedLength,
    print('The surcharge for service 0120 can only be charged once per treatment day.'),
    true.
```

2220:

```
is_invoice_invalid(Invoice) :-
    Invoice = invoice(Services, _, _, _),
    findall(Session, (member(service(Service, Session, _, _, _, _, _, _), Services), between(2050,
           2130, Service)), Sessions1),
    findall(Session, (member(service(Service, Session, _, _, _, _, _, _), Services), between(2200,
           2220, Service)), Sessions2),
    intersection(Sessions1, Sessions2, CommonSessions),
    CommonSessions \= [],
    print('Services 2200 to 2220 charged alongside services 2050 to 2130.'),
    true.
is_invoice_invalid(Invoice) :-
    Invoice = invoice(Services, _, _, _),
    member(service(2210, _, _, _, _, Description, Justification, _), Services),
    (sub_string(Description, _, _, _, 'implant'); sub_string(Justification, _, _, _, 'implant')),
    print('Service 2210 charged in connection with implants.'),
    true.
is_invoice_invalid(Invoice) :-
    Invoice = invoice(Services, _, _, _),
    member(service(Service, _, _, _, _, _, Justification, _), Services),
    member(Service, [2230, 2240]),
    \+(sub_string(Justification, _, _, _, 'impossible')),
    \+(sub_string(Justification, _, _, _, 'not indicated')),
    print('Services 2230 and 2240 charged without proper justification.'),
    true.
```

A.3.5 Example invoices for testing rules

0120:


```
{
    "is_correct": true,
    "tests_rule": "0120",
    "explanation": "The surcharge according to number 0120 amounts to 100% of the basic fee rate for
           the respective service, but not more than 68 euros. The surcharge according to number 0120
           can only be calculated once per treatment day.",
    "invoice": {
      "_Positionen": [
        {
          "_Faktor": 2.3,
          "_Behandlungsdatum": "2012-02-12",
          "_Anzahl": 1,
          "_Betrag": 50.71,
          "_Leistungsbeschreibung": "Aufbereitung eines Wurzelkanals",
          "_Nr": "2410",
          "_Typ": "Service"
        },
        {
          "_Faktor": 1.0,
          "_Behandlungsdatum": "2012-02-12",
          "_Anzahl": 1,
          "_Betrag": 50.71,
          "_Leistungsbeschreibung": "Zuschlag  f  r  die Anwendung eines Lasers bei Leistung nach Nummer
                 2410",
          "_Nr": "0120",
          "_Typ": "Service"
        },
        {
          "_Faktor": 2.3,
          "_Behandlungsdatum": "2012-03-12",
          "_Anzahl": 1,
          "_Betrag": 50.71,
          "_Leistungsbeschreibung": "Aufbereitung eines Wurzelkanals",
          "_Nr": "2410",
          "_Typ": "Service"
        },
        {
          "_Faktor": 1.0,
          "_Behandlungsdatum": "2012-03-12",
          "_Anzahl": 1,
          "_Betrag": 50.71,
          "_Leistungsbeschreibung": "Zuschlag  f  r  die Anwendung eines Lasers bei Leistung nach Nummer
                 2410",
          "_Nr": "0120",
          "_Typ": "Service"
        }
      ],
      "_Kassenanteil": {
        "_KassenanteilSumme": null
      },
      "_Rechnungsnummer": "97997",
      "_Rechnungsdatum": "2012-07-03",
      "_Rechnungsbetrag": 202.84,
      "_Waehrung": "EUR"
    }
  },
  {
    "is_correct": false,
    "tests_rule": "0120",
    "explanation": "The surcharge according to number 0120 amounts to 100% of the basic fee rate for
           the respective service, but not more than 68 euros. The surcharge according to number 0120
           can only be calculated once per treatment day.",
    "invoice": {
      "_Positionen": [
```

```
      {
        "_Faktor": 2.3,
        "_Behandlungsdatum": "2012-02-12",
        "_Anzahl": 1,
        "_Betrag": 50.71,
        "_Leistungsbeschreibung": "Aufbereitung eines Wurzelkanals",
        "_Nr": "2410",
        "_Typ": "Service"
      },
      {
        "_Faktor": 1.0,
        "_Behandlungsdatum": "2012-02-12",
        "_Anzahl": 1,
        "_Betrag": 50.71,
        "_Leistungsbeschreibung": "Zuschlag f r die Anwendung eines Lasers bei Leistung nach Nummer
               2410",
        "_Nr": "0120",
        "_Typ": "Service"
      },
      {
        "_Faktor": 1.0,
        "_Behandlungsdatum": "2012-02-12",
        "_Anzahl": 1,
        "_Betrag": 50.71,
        "_Leistungsbeschreibung": "Zuschlag f r die Anwendung eines Lasers bei Leistung nach Nummer
               2410",
        "_Nr": "0120",
        "_Typ": "Service"
      }
    ],
    "_Kassenanteil": {
      "_KassenanteilSumme": null
    },
    "_Rechnungsnummer": "97997",
    "_Rechnungsdatum": "2012-07-03",
    "_Rechnungsbetrag": 152.13,
    "_Waehrung": "EUR"
  }
},
{
  "is_correct": false,
  "tests_rule": "0120",
  "explanation": "The surcharge according to number 0120 amounts to 100% of the basic fee rate for
         the respective service, but not more than 68 euros. The surcharge according to number 0120
         can only be calculated once per treatment day.",
  "invoice": {
    "_Positionen": [
      {
        "_Faktor": 2.3,
        "_Behandlungsdatum": "2012-02-12",
        "_Anzahl": 1,
        "_Betrag": 50.71,
        "_Leistungsbeschreibung": "Aufbereitung eines Wurzelkanals",
        "_Nr": "2410",
        "_Typ": "Service"
      },
      {
        "_Faktor": 1.0,
        "_Behandlungsdatum": "2012-02-12",
        "_Anzahl": 1,
        "_Betrag": 70.00,
        "_Leistungsbeschreibung": "Zuschlag f r die Anwendung eines Lasers bei Leistung nach Nummer
               2410",
        "_Nr": "0120",
        "_Typ": "Service"
      }
    ],
```

```
      "_Kassenanteil": {
        "_KassenanteilSumme": null
      },
      "_Rechnungsnummer": "97997",
      "_Rechnungsdatum": "2012-07-03",
      "_Rechnungsbetrag": 120.71,
      "_Waehrung": "EUR"
    }
  }
```


2220:


```
{
    "is_correct": true,
    "tests_rule": "2220",
    "explanation": "",
    "invoice": {
      "_Positionen": [
        {
          "_Faktor": 2.3,
          "_Behandlungsdatum": "2012-02-12",
          "_Anzahl": 1,
          "_Betrag": 267.38,
          "_Leistungsbeschreibung": "Versorgung eines Zahnes durch eine Teilkrone",
          "_Nr": "2220",
          "_Zahn": [11],
          "_Typ": "Service"
        },
        {
          "_Faktor": 2.3,
          "_Behandlungsdatum": "2012-02-12",
          "_Anzahl": 1,
          "_Betrag": 147.60,
          "_Leistungsbeschreibung": "Einlagefllung, einfl chig",
          "_Nr": "2150",
          "_Zahn": [11],
          "_Typ": "Service"
        }
      ],
      "_Kassenanteil": {
        "_KassenanteilSumme": null
      },
      "_Rechnungsnummer": "97997",
      "_Rechnungsdatum": "2012-07-03",
      "_Rechnungsbetrag": 414.98,
      "_Waehrung": "EUR"
    }
  },
  {
    "is_correct": false,
    "tests_rule": "2220",
    "explanation": "Neben den Leistungen nach den Nummern 2200 bis 2220 sind die Leistungen nach den
          Nummern 2050 bis 2130 nicht berechnungsfhig.",
    "invoice": {
      "_Positionen": [
        {
          "_Faktor": 2.3,
          "_Behandlungsdatum": "2012-02-12",
          "_Anzahl": 1,
          "_Betrag": 267.38,
          "_Leistungsbeschreibung": "Versorgung eines Zahnes durch eine Teilkrone",
          "_Nr": "2220",
          "_Zahn": [11],
          "_Typ": "Service"
        },
        {
          "_Faktor": 2.3,
          "_Behandlungsdatum": "2012-02-12",
```

```
        "_Anzahl": 1,
        "_Betrag": 27.55,
        "_Leistungsbeschreibung": "Pr parieren einer Kavit t und Restauration mit plastischem
               F llungsmaterial",
        "_Nr": "2050",
        "_Zahn": [11],
        "_Typ": "Service"
      }
    ],
    "_Kassenanteil": {
      "_KassenanteilSumme": null
    },
    "_Rechnungsnummer": "97997",
    "_Rechnungsdatum": "2012-07-03",
    "_Rechnungsbetrag": 294.93,
    "_Waehrung": "EUR"
  }
},
{
  "is_correct": false,
  "tests_rule": "2220",
  "explanation": "Neben den Leistungen nach den Nummern 2200 bis 2220 sind die Leistungen nach den
         Nummern 2050 bis 2130 nicht berechnungsf hig.",
  "invoice": {
    "_Positionen": [
      {
        "_Faktor": 2.3,
        "_Behandlungsdatum": "2012-02-12",
        "_Anzahl": 1,
        "_Betrag": 217.06,
        "_Leistungsbeschreibung": "Versorgung eines Zahnes durch eine Vollkrone",
        "_Nr": "2210",
        "_Zahn": [11],
        "_Typ": "Service"
      },
      {
        "_Faktor": 2.3,
        "_Behandlungsdatum": "2012-02-12",
        "_Anzahl": 1,
        "_Betrag": 83.05,
        "_Leistungsbeschreibung": "Pr parieren einer Kavit t und Restauration mit
               Kompositmaterialien",
        "_Nr": "2100",
        "_Zahn": [11],
        "_Typ": "Service"
      }
    ],
    "_Kassenanteil": {
      "_KassenanteilSumme": null
    },
    "_Rechnungsnummer": "97997",
    "_Rechnungsdatum": "2012-07-03",
    "_Rechnungsbetrag": 300.11,
    "_Waehrung": "EUR"
  }
}
```