

Chain of Prompts and Memories for Accessible Symbolic Applications in the Legal Domain

Marco Billi, Alessandro Parenti, Giuseppe Pisano and Marco Sanchi

University of Bologna, Italy

Abstract

This article introduces a novel hybrid approach that enhances the accessibility and explainability of symbolic legal systems by integrating them with large language models (LLMs). While symbolic systems provide transparency and justifiability, they suffer from communicability issues and limitations in knowledge representation. Conversely, LLMs offer natural language fluency but struggle with reliability and justifiability in legal reasoning. The article bridges this gap by structuring legal tasks into a modular, interconnected sequence of prompts, supported by contextually relevant memory retrieval, thus creating a chain of prompts and memories. The methodology is validated through two case studies. The first applies it to computable contracts, demonstrating its ability to map code to contractual provisions, thereby making them more accessible. The second case study involves an expert legal system, where the hybrid approach generates accessible explanations of Prolog-based legal reasoning. The findings show that the hybrid approach enhances the clarity and interpretability of legal applications while preserving justifiability, using a modular structure that allows for adaptability across different legal tasks.

Keywords: Large language models; rule-based systems; symbolic legal systems; computable contracts; accessibility; chain of prompts and memories.

1. Introduction¹

The field of law is increasingly embracing digital technologies, with symbolic legal systems playing an established role in automating legal reasoning and juridical tasks across various domains. Such systems achieve their goal by representing expert legal knowledge in a formal and computable way using high-level programming languages, contributing to the creation of a variety of applications.

Among them, this article focuses on *symbolic legal reasoners* and *computable contracts* as legal informatics applications that have found successful use in a variety of socio-juridical domains, such as credit, insurance, welfare and pensions.²

First, a clear and well-known example of a symbolic system used for automated reasoning is represented by rule-based systems. These models ground their functioning on the application of an expert knowledge base made of rules, relationships and statements, which are applied through an inference engine to factual data related to specific cases, enabling transparent and justifiable reasoning.³

¹ All authors contributed equally to this work.

² Lymer, “Hybrid-based Expert System”; Michaelsen, “Expert System”; Masri, “Survey of Rule-based Systems”; van Melle, “Mycin”; Masson, “Comprehension of Legal Contracts”; Armitage, “Trust, Confidence.”

³ Ligeza, “Logical Foundations.”



On the second end, *computable contracts* are legal agreements that are expressed in whole or in part as computer programs.⁴ These systems usually consist of a stack of multiple representations of the same agreement expressed through different languages,⁵ including natural language and programming languages. The employment of software in this context can provide benefits by reducing transaction costs related to the contractual relationship. Provided that some terms of the agreement can be expressed as computer-readable data, computer programs can automate performances, compliance-check and streamline the monitoring of the contract life-cycle.

While symbolic models offer advantages in terms of justifiability and trustworthiness, they nevertheless carry a well-established and documented set of problems. Often being presented in high-level programming languages, in fact both the functioning and output of such systems can be challenging to understand and appreciate by laypeople, due to their technical and expert nature.⁶ Moreover, due to their predetermined nature and scope, symbolic systems struggle to capture the nuances, exceptions and ambiguities inherent in the law and legal language, which are difficult to encode in a way that allows for agile and cost-effective updates.⁷ This inflexibility hinders the ability of legal expert systems to accurately represent real-world legal scenarios, limiting their utility for laypeople.

Considering these drawbacks, a different approach and branch of legal informatics are developing around the use of large language models (LLMs), a type of generative artificial intelligence system that leverages deep learning methodologies to achieve the processing and creation of text in both natural and programming languages. Such models have been the subject of research in the legal domain, with systems being tested on tasks such as retrieval of legal information, contract drafting, artificial legal counsel and legal writing, including simplification of legal documents and generation of explanations.⁸ Although they are promising, concerns exist about the reliability and justifiability of their outputs, particularly in the context of complex legal tasks. This is due to the intrinsically opaque nature of their reasoning, their tendency to misinterpret specific contextual information and the complications caused by the phenomenon of hallucinations.⁹ Above all, both LLMs and symbolic legal systems grapple with distinct yet significant challenges and weaknesses, which challenge the trustworthiness of the former and the accessibility of the latter when used for the automation of legal tasks.

To overcome these limitations, this article develops a novel *hybrid approach* that combines the explainable, justifiable and trustworthy nature of symbolic legal systems with the natural language processing capabilities of LLMs, in an effort to augment symbolic applications by enhancing their accessibility, versatility and utility. By leveraging the dynamic adaptability of LLMs, this hybrid approach aims to provide not only static simplifications but also interactive, context-aware explanations that address different user needs and backgrounds.

To achieve such a goal, this research develops the chain of prompts (CoP) approach,¹⁰ a modular and versatile chain of operations where each input builds off of the previous output and establishes the premises for the next. We obtain a continuous sequence of operations where each link represents a legal task, which can be taken as a singular instance in a modular fashion, or as part of a more complex stream of operations, culminating in the final output.

The study continues by implementing memory through retrieval augmented generation (RAG) in the chain. This addition has the goal of giving precise and relevant context to each of the chained prompts by offering information suitable to carry out the specific legal task, ensuring that each step has direct references to formal legal data.¹¹

This all creates a set of memories that not only follows and augments the CoP but, in a similar fashion, combines the memory from previous tasks to enhance the next, thus developing an underlying chain of memories (CoM). This study develops an original methodology based on the integration of the two chains, achieving what is here defined as a chain of prompts and memories (CoPM) (see Figure 1 later in the article). Symbolic reasoning supports the CoPM by ensuring its justifiability. The output of the overall hybrid system is explained through the tracing of each step of the chain, with the first step firmly grounded in formal legal information. Moreover, the justifiability of the architecture enables users to verify all the results generated

⁴ Surden, “Computable Contracts.”

⁵ Allen, “Wrapped and Stacked.”

⁶ Susskind, “Expert Systems in Law”; Schauer “Playing by the Rules.”

⁷ Governatori, “Thirty Years of Artificial Intelligence.”

⁸ Blair-Stanek, “Can GPT-3 Perform?”; Trautmann, “Legal Prompt Engineering”; Sun, “A Short Survey”; Deroy, “Ensemble Methods.”

⁹ Savelka, “Explaining Legal Concepts.”

¹⁰ Trautmann, “Large Language Model Prompt Chaining”; Billi, “Accessible Rule-based Reasoning”; Billi, “Generative Symbolic Approach.”

¹¹ Martineau, “What is Retrieval-Augmented Generation?”

through the reference of symbolic rules and contextual information present within the hybrid system, validating each instance of the chain.¹²

The CoPM is tested on two case studies, which follow in an effort to demonstrate the practical effectiveness of the hybrid approach and support its validation. The first case study addresses computable contracts written in the Stipula programming language¹³ (Section 4.1). A description of the code is provided, together with a detailed explanation of its legal meaning in relation to the natural language agreement, in an effort to bridge the two representations. The second case study concerns the rule-based reasoner CrossJustice (Section 4.2). Here, a natural language translation and an explanation of the system output – originally given in the Prolog high-level language¹⁴ – is achieved, showing how the hybrid approach can be applied to declarative and procedural languages alike.¹⁵

2. Related Works

Among technical studies supporting laypeople in understanding legal language and reasoning¹⁶ is one that explores the potential of generative AI to bridge the access to justice gap for low-income litigants in the United States. Drawing on user cases from Arizona courts, the study explores case studies to highlight the practical benefits of LLMs for the law, which includes making legal processes more accessible by improving the understandability of complex legal documents and legal information for laypeople. Continuing, the study¹⁷ investigates the role of generative AI in improving access to justice through a field study and survey, showing how LLMs can improve legal language accessibility and understanding within legal aid initiatives. Specifically, between a variety of AI use cases, legal aid professionals participating in the study applied LLMs for the translation of legalese into more accessible forms of legal language communication.

In this regard, efforts to improve the comprehensibility of domain language do not appear novel and can be traced back to the plain language movement of the 1960s,¹⁸ which sought to simplify legal documents for broader layperson accessibility. More recently, approaches such as layperson ontologies¹⁹ have been explored to enhance the retrieval and presentation of legal information in ways that align with non-expert users' conceptual understanding. Additionally, Natural Language Processing (NLP) techniques have been employed to simplify legal documents and statutes,²⁰ making legal texts more accessible through automated lexical and syntactic modifications. Our work contributes to this body of knowledge, focusing on formal languages for symbolic legal AI applications. The simplification of legal language is the first step towards a complex framework enhancing the interaction between symbolic systems and laypeople.

On the topic of legal information presentation and simplification, the accessibility of expert legal systems appears tied to the understandability of their output, particularly considering its specific programming language.²¹ Kowalski has investigated ways to make the programming languages used for symbolic legal reasoning accessible to laypeople. One such approach involves the development of Controlled Natural Languages (CNLs) that maintain computational rigor while approximating human language in structure and readability. A prominent example in the legal domain is logical English,²² which is designed to bridge the gap between natural language and formal rule-based representations. By maintaining a syntax that resembles English, this approach improves the intelligibility of legal rules for both users and developers. While these works demonstrate interest from the legal community in understanding and developing programming languages that are accessible to end-users, they do not incorporate LLMs into their design. Our pipeline does so by leveraging LLMs to translate high-level programming languages, overcoming their fixed and static nature through the versatility of natural language generation.

¹² Mittelstadt, "Explaining Explanations in AI"; Doshi-Velez "Accountability of AI Under the Law."

¹³ Crafa, "Pacta sunt servanda."

¹⁴ Billi, "Explainability through Argumentation in Logic Programming."

¹⁵ Declarative languages focus on *what* the program should accomplish – that is, describing the desired result (e.g., Prolog), while procedural languages specify *how* to accomplish it through step-by-step instructions (e.g., Python).

¹⁶ Chien, "Access to Justice Gap."

¹⁷ Chien, "Generative AI and Legal Aid."

¹⁸ Mellinkoff, "The Language of the Law"; Charrow, "Making Legal Language Understandable."

¹⁹ Uijttenbroek, "Retrieval of Case Law"; Barrera, "From User Needs."

²⁰ Paquin, "Loge-Expert"; Garimella, "Text Simplification."

²¹ Morris, "Blawx"; Merigoux, "Catala."

²² Kowalski, "Logical English."

In line with this effort,²³ Nguyen bridges the gap between legal expertise and programming by integrating logic programming with LLMs. Similar works have also applied the same principles to the Prolog language, such as²⁴ showing that LLMs understand Prolog syntax and are able to replicate it in natural language, with the first work focusing on accessibility purposes. In this regard, Westermann²⁵ presents a tool that exemplifies a methodology for creating systems that empower laypeople and increase access to justice. Finally, an additional example of LLM and expert system interaction for law and technology is shown by the work of Zin,²⁶ where generative AI is applied to the automated extraction of traffic normative rules from judicial decisions. These contributions specifically address how generative AI can be leveraged to interpret legal language, normative elements and logic programming, enhancing the representation of legal rules and symbolic reasoning. Our work expands on the interaction between symbolic systems and LLMs, contributing to the development of hybrid architectures through RAG and innovative prompting strategies (CoPM).

3. Chain of Prompts and Memories

We now present our methodology for the augmentation of symbolic models through the deployment of LLMs.

We introduce what we refer to as the chain of prompts approach (Figure 1), a technique that builds on the principles of the chain of thought²⁷ method to create a complex, yet flexible architecture of interconnected sets of inputs and outputs. In the chain of prompts approach, each prompt governs a specific legal task, such as comparing different legal sources, identifying rule similarities or retrieving relevant legal information, which are carried out by the LLM, referencing the formalised legal knowledge of the symbolic system in a hybrid way. Each prompt is developed and deployed as part of a series of interconnected steps and operations, with the output of the first acting as the input to the second, and so forth, establishing a complex chain of inputs and outputs. The result is a modular design that creates a versatile sequence of automatic legal tasks, with each prompt building upon the output of the previous one and establishing the foundation for the next step.

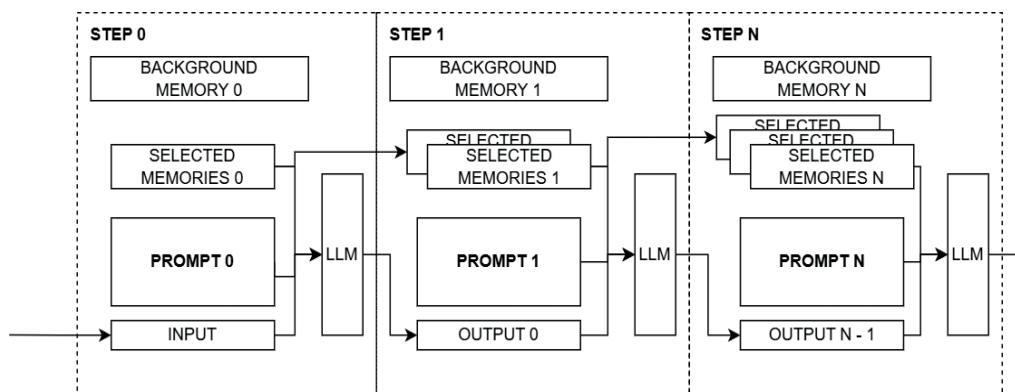


Figure 1. Chain of prompts and memories

The final outcome is an uninterrupted series of operations where each CoP link represents a distinct legal task that can be executed independently in a modular fashion, or integrated into a more intricate workflow, ultimately leading to the desired final output. This modular structure empowers flexibility and adaptability in addressing the diverse legal needs and tasks that users may request from the system, given that the chain can be interrupted at any point, meaning at any task, without any prejudice to the complex overall architecture.

To bolster the stability of the CoP framework, we incorporate background memory into the chain itself by implementing a Retrieval Augmented Generation (RAG) architecture into our pipeline²⁸. RAG represents a recent advancement in LLMs that addresses limitations associated with static training data. The approach integrates external knowledge sources, such as databases or domain-specific repositories, during the generation process to boost the accuracy of the system and limit the chance of

²³ Nguyen, “Beyond Logic Programming”; Nguyen, “Lawgiba.”

²⁴ Billi, “Accessible Rule-based Reasoning”; Billi, “Fighting the Knowledge Representation Bottleneck.”

²⁵ Westermann, “Bridging the Gap.”

²⁶ Zin, “LLM for Identification.”

²⁷ Wei, “Chain-of-Thought Prompting”; Kojima, “Large Language Models.”

²⁸ Gao, “Retrieval-Augmented Generation.”

hallucinations. This allows the model to access factual information and contextualise its responses, leading to improved reliability, informativeness and task-relevance.

The implementation of context supports each prompt within the CoP with precise and relevant knowledge, thereby strengthening each link of the chain and the respective operation. Specifically, the goal is to provide contextual information tailored to each legal task, ensuring that every step has clear and direct references to formal legal information, such as relevant case law, examples of contracts and regulation. To achieve this, we first divide the relevant information into distinct memory units that serve as specific contextual backgrounds for individual prompts and tasks. A further step consists of combining various relevant memories into a broader, yet still specific and focused, context to address complex legal tasks.

Such a feature is fundamental to our hybrid approach, as it directly tackles a characteristic inherent in the way RAG functions in practical terms. Specifically, memory retrieval hinges on the similarity between the prompt and the text, with the generative model using the prompt as input to identify relevant information, which subsequently implies that poorly formulated prompts with weak semantic content may not yield any useful retrieval results. However, the modular chained architecture implemented in our hybrid approach addresses and supports this feature. In practice, the CoPM architecture creates a flow where, for a given prompt and corresponding legal task, context and memory are inherited from the previous prompt and coordinate with the present one, allowing us to be free of engineering the following prompt of the chain in the most appropriate way, regardless of its semantic richness. In other words, the CoPM architecture achieves freedom of operation for experts who engage with the system, given how context and information flow through the chain of interconnected legal operations.

Our approach lays its foundations on the grounded, stable and accurate knowledge of a symbolic legal system, which acts as the primary source of justifiability of the resulting combined architecture, where the LLM executes tasks based either directly on symbolic knowledge, or indirectly through the chaining of outputs based on it. Continuing, memory further supports justifiability by offering information that is inherently relevant for a given task, as Retrieval Augmented Generation applications allow us to verify specifically which pieces of memory have been used to generate a response, recalling legal knowledge directly.

It follows that, through the referencing of expert symbolic knowledge, the chaining of inputs and outputs based on it, and the presence of recalled legal information through memory, the CoPM approach creates an architecture that presents responses which can be validated and verified, realising an augmented, yet still justifiable system.

3.1 Evaluation Criteria

State-of-the-art literature on the evaluation of LLM performance suggests employing both quantitative and qualitative evaluations to validate its output in the context of NLG and NLP tasks.²⁹ It argues for the deployment of expert evaluators and human-driven assessment criteria alongside traditional, automatic validation metrics to offer holistic and comprehensive performance reviews.³⁰ Such a view is supported in the legal domain, where research on LLM performance evaluation accounts for specificities in the nature of legal tasks to argue for the need to include human experts in the assessment loop.³¹ For instance, standard metrics based on the IRAC framework³² emerge, suggesting their suitability to evaluate generative legal reasoning. Moreover, technical studies exploring the interaction, support and application of generative AI to symbolic legal systems reflect this, showing the inclusion of human evaluators in the task assessment loop as domain experts.³³

To validate the results of its case studies, this article expands on the research presented above to introduce a human-driven, qualitative expert validation. The LLM performance evaluation is carried out by human experts relying on assessment criteria adapted and derived from the research conducted by Chang.³⁴ The criteria are described below and have the goal of assessing not only the interaction between the generative and symbolic systems, but also and most importantly the accessibility, justifiability and legal soundness of the final output. Considering the nature of outputs deriving from the case studies as unstructured outcomes based on legal language generation, as well as a lack of pre-existing elements that could be used as benchmarking standards, this research does not employ quantitative, automatic evaluation metrics and scores, relying solely on the qualitative assessment carried out by legal and IT experts.

²⁹ Chang, “A Survey on Evaluation of LLMs”; Sottana, “Evaluation Metrics.”

³⁰ Liang, “Holistic Evaluation.”

³¹ Sanchi, “Towards Hybrid Evaluation Methodologies.”

³² Katz, “Gpt-4 Passes the Bar Exam.”

³³ Janatian, “From Text to Structure”; Billi, “Fighting the Knowledge Representation Bottleneck.”

³⁴ Chang, “A Survey on Evaluation of LLMs.”

The qualitative evaluation criteria are:

- *Correctness*. Accuracy in grasping key points and essential information by the model. This criterion excludes outputs that misrepresent the meaning and argumentation of the legal programming language, ensuring proper interpretation and application of rules.
- *Form*. Correspondence between input and output in terms of structure and presentation. This criterion ensures coherence, readability and accessibility of the output for laypeople in a given legal task, such as the simplification of legal vocabulary (legalese), under a lens of formal validation.
- *Completeness*. Inclusion of all elements requested by the prompt, with a particular emphasis on those necessary to determine whether the operation is successful. This criterion highlights instances where outputs fail to account for case-specific facts linked to the symbolic knowledge base, under a lens of substantial validation.

3.2 Technological Setup

We have chosen GPT-4o³⁵ as our LLM, as it constantly ranks among the top models for legal tasks according to several benchmarks.³⁶ To implement our RAG pipeline, we leveraged the Cheshire Cat AI framework,³⁷ a versatile system that seamlessly integrates retrieval and generation components. The framework is designed for memory-augmented applications, with a long-term memory system composed of three key components: episodic memory, which stores extracts of past user interactions; declarative memory, which retains extracts of uploaded documents; and procedural memory, which consists of Python functions that enhance and extend the framework's generation capabilities. For our use case, we exclusively relied on declarative memory, as our focus was on structured knowledge retrieval rather than chatbot-style interactions. We completely disabled episodic memory and procedural memory.

The embedding model used was text-embedding-ada-002,³⁸ a highly efficient encoder optimised for multilingual text with strong semantic understanding. Similarity searches were performed using cosine similarity, which measures the closeness of two vectors based on the angle between them. This approach ensures that retrieval captures semantic meaning rather than relying solely on exact keyword matches, improving the accuracy and relevance of responses.

More details about the technological setup, along with the source code and all relevant datasets, can be found in the project repository.³⁹

4. CoPM for Accessibility

In this section, we present the way LLMs can facilitate users' understanding of symbolic technologies, enhancing their accessibility.

4.1 Computable Contracts – *Stipula*

We will apply the CoPM methodology to computable contracts – legal agreements that are expressed in whole or in part as computer programs and that are provided with the means to automatically enforce some of the obligations they define (Figure 2).

³⁵ <https://platform.openai.com/docs/models/gpt-4o>.

³⁶ <https://www.vals.ai/home>.

³⁷ <https://cheshirecat.ai/>.

³⁸ <https://platform.openai.com/docs/guides/text>.

³⁹ <https://github.com/copmAI/copm>.

Loan Agreement

...

2. Repayment

Subject to the other terms of this agreement, Borrower will repay the loan in the following payments:

- (a) Payment 1, due June 1, 2023, in the amount of \$550, representing a payment of \$500 as half of the principal and interest in the amount of \$50.
- (b) Payment 2, due June 1, 2024, in the amount of \$525, representing a payment of \$500 as the remaining half of the principal and interest in the amount of \$25.

3. Events of Default

The Borrower will be in default under this agreement upon the occurrence of any of the following events or conditions, provided they shall remain uncured within a period of two days after notice is given to Borrower by Lender of their occurrence (such an uncured event an “Event of Default”):

- (a) Borrower shall fail to make timely payment of any amount due to Lender hereunder; A default will be cured by the Borrower (i) remedying the potential event of default and (ii) giving effective notice of such remedy to the Lender.

...

Figure 2. Loan agreement

To encode contracts, we will employ the Stipula⁴⁰ programming language – an open-source domain-specific procedural language designed specifically for legal contracts.

Stipula is built on top of a small set of primitives that are designed to reflect the main basic elements of legal agreements:

- *States* are used to implement prohibitions and permissions. Stipula programs are defined as a finite-state machine where each state represents a stage of the contract life cycle. States (indicated with an ‘@’ in front) can be used to enable or disable the invocation of functions.
- *Events* are used to implement contract obligations. These constructs define a timer for triggering specific operations, provided that certain conditions are met. They can be employed to check the fulfilment of obligations and trigger specific consequences.
- *Functions* represent the actions that can be performed by parties. They define the party that can call the function and the state when the calling is admitted. Their invocation can be conditioned by specific parameter constraints indicated in brackets.

The full Stipula language also includes other primitives such as the Agreement constructor, assets or higher-order functions to implement amendments. For the specific syntax and semantics of the full version of Stipula, see the work of Crafa⁴¹ and Laneve.⁴²

⁴⁰ Crafa, “Pacta Sunt Servanda.”

⁴¹ Crafa, “Pacta Sunt Servanda.”

⁴² Laneve, “Legal Contracts Amending”; Laneve “Draft Better Contracts.”

For the present scenario, we employ a contract example from Flood.⁴³ For reasons of space, throughout this article we focus on just one part of the contract that is shown in Figure 2.⁴⁴ The contract defines a loan agreement between a lender and a borrower, where the lender commits to transfer a principal amount of \$1000 with a 5 per cent annual interest rate. Parties also define a precise schedule for repayment along two years (article 2), and the rules to be applied in case the payment obligations are not met in due time (article 3). The contract provisions of Table 1 can be implemented in Stipula as shown in Figure 3.

```

1  @Q4 Borrower: f3() [a] (a == 550) {
2    a -o Lender
3    a -> field4;
4    -
5  } ==> @Q5
6
7  @Q4 Lender: fn() [] (field3 > 0) {
8    "notice" -> Borrower;
9    now + 48 >> @Q4 {default -> Borrower} ==> @Qd
10 } ==> @Q4

```

Figure 3. Loan agreement in Stipula

This snippet of the computable contract shows two functions: the first (f3) enables the first repayment by the borrower according to article 2(a). The second (fn) allows the lender to issue a notice in case of potential default according to article 3. Both functions can be called in state @Q4, but they feature different constraints (lines 1 and 7 in brackets).

Grasping the connection between the natural language contract and its representation in a programming language, and understanding how one technically implements the other, are not trivial. These are highly specialised tasks that require expert knowledge and proficiency in both programming and private law. It becomes even more challenging when the computable representation is written in a procedural programming language (in contrast to the case in Section 4.2), as there is not always a direct one-to-one correspondence between contract terms and code. These languages often use structures that serve a purpose in the program's logic but may not directly resemble any part of the contract. In such context, interpreting the code logic in light of the contract is particularly challenging. We will show how the link between these two representations can be captured and presented to the users in an accessible way through the use of LLMs and CoPM.

The aim is to provide the user of a Stipula contract with the necessary information to understand how the logic of the code was defined to implement specific provisions of the agreement. The achievement of such a goal involves three main requirements that have to be satisfied by the language model:

1. grasping the code's meaning in terms of its syntax and semantics (*technical meaning*)
2. identifying the specific provision of the contract that each piece of code intends to implement (*legal meaning*)
3. presenting this information in a clear and accessible way.

To achieve this, we initially fed the full contract code together with the natural language agreement and prompted the model to provide the information just described. However, we soon realized that this goal could not be achieved through such an approach. First, answers were very unreliable as to their completeness, often overlooking either some pieces of code that did not have an evident correlation with a contract provision or provisions that are not implemented in code. Moreover, the system struggled to accurately interpret the semantics of the programming language. This issue was particularly evident with Stipula's event-based syntax. For example, the statement:

now + 48H >> @Q4{...} ==> @Qd

means that after a 48-hour timer expires, the system will check whether the contract is in state @Q4, and only then, if that condition is met, it will transition to state @Qd.

In contrast, the model interpreted the runtime behaviour as follows: 'If, during that period [the 48 hours], the Borrower executes a valid payment action (like f3), the payment is recorded, and field3 becomes zero or non-positive. If no such action occurs, and field3 is still > 0 after 48 hours, the transition to @Qd (default state) is triggered automatically.'

⁴³ Flood, "Contract as Automation."

⁴⁴ You can find the full text of the contract together with the complete version of the code at <https://github.com/copmAI/copm/blob/main/res/contract.md>.

This interpretation incorrectly assumes that the condition for the transition is based on the value of field3 (used as a constraint of the fn function – see Figure 3), rather than on the contract’s current state. This is because the system does not know the language and therefore guesses its meaning based only on the syntax provided.

Acknowledging these issues, we switched to the CoPM methodology. As shown in Figure 4, we first retrieved the targeted piece of code from the full program (Section 4.1.1), then we extracted its technical meaning (Section 4.1.2) and finally, we processed our results together with the natural language contract to identify the connection between contract and code (Section 4.1.3).

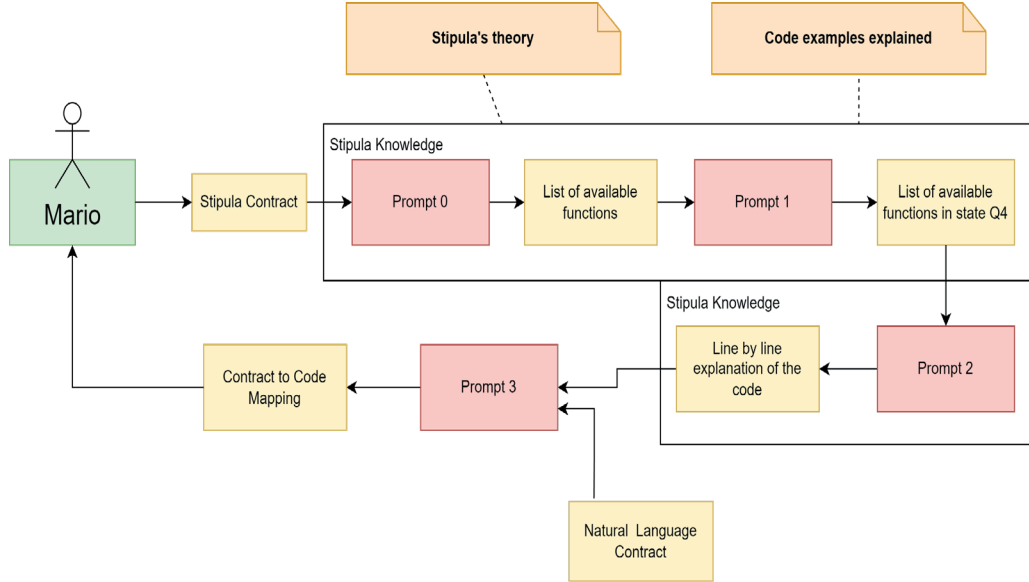


Figure 4. CoPM – deciphering computable contracts

One of the main challenges was to employ a programming language not known by the language model in use. To address this, we exploited the RAG architecture to feed the model the necessary knowledge about the Stipula language. The data uploaded originally consisted of parts of scientific papers describing the background theory of the language, including motivations and design concepts at its basis. We included the description of the main primitives composing Stipula’s syntax and, finally, some examples of Stipula contracts with a related explanation of their code.

At first, we uploaded this information in three separate text documents. However, through our experiments we observed that large documents covering multiple topics prevented the system from using the provided knowledge effectively. The model often included unrelated information in its responses, likely because relevant and irrelevant details were retrieved from the same memory segment, leading to confusion and errors. For example, when explaining the instruction `a-o Lender`, which refers to a payment of an instalment made by the borrower, the model retrieved a memory segment describing a different Stipula contract example. Although that example also involved a lender and a borrower, it referred to a locker rental agreement, and therefore returned: ‘the token is sent back to the Lender, who regains full control of the locker’.

Indeed, providing full examples alongside their complete line-by-line explanations made it difficult for the system to correctly match each explanation with the corresponding piece of code. To address these issues, we reorganised the information into smaller, more specialised documents. Each document either focused on a single aspect of the language, such as a specific syntactic primitive, or paired a short code snippet with its corresponding explanation.

4.1.1 Extracting Available Functions

The goal of first step in the pipeline is to narrow down the scope of the analysis, starting from the full code of the computable contract. Assuming that the contractual relationship described by the agreement in Figure 2 has reached the stage where the first repayment of the loan becomes due, the contract’s user wants to understand what actions can be performed through the computable contract and what their effect is in relation to the natural language agreement. The phase just described is represented by state @Q4 in the Stipula code of Figure 3.

We fed the LLM the full Stipula code and tasked it with extracting all the functions that can be invoked from state @Q4. In Stipula’s syntax, the state allowing the invocation of a function is indicated at the beginning of its body, right before the invoking party. Nevertheless, the LLM initially misinterpreted the code returning also functions in which the state @Q4 generally appeared within its body. When asked to explain its answers, the system would recognise the mistake, explaining how the function was actually not callable from that state. Moreover, the chunks of documents retrieved from the augmented memory were the most relevant ones, showing exactly the meaning of events by examples. While the system shows a good knowledge of the programming language when asked, directly processing large amounts of data without being constrained by the prompt can nevertheless produce errors in the output.

In order to solve this issue, we split this task in two minor ones implementing the CoPM methodology further. As a first step, we fed the LLM the full contract code together with the prompt shown in Figure 5.

```

1 You will receive a contract written in the stipula programming language.
2 - identify all the functions that show the state @Q4 in their whole body;
3 - return a list of all functions found and print the referenced piece of code.
4
5 Follow the following structure:
6 - function name: piece of code;

```

Figure 5. Prompt 0 (deciphering computable contracts)

We adapted to the system’s reasoning and simply required it to retrieve all the functions including ‘@Q4’ in their body. The LLM easily fulfilled this task and generated the list of requested functions. Through this procedure, we reduced the dataset size for our analysis.

We fed the output just obtained in the next prompt together with the instructions shown in Figure 6.

```

1 You will receive a list of functions written in the stipula programming language.
2 1. for each function, identify what party can invoke the function;
3 2. for each function, identify in what state the function can be invoked;
4 3. print the results
5 4. according to step 3, discard all the functions that cannot be called in state @Q4.
6 5. print the piece of code of the functions remaining from step 4.

```

Figure 6. Prompt 1 (deciphering computable contracts)

The prompt outlines a step-by-step analysis that has to be carried out on the piece of code inputted. This would allow the LLM to extract the necessary information from the code (lines 2–3) and then to make an evaluation on that basis (line 4).

In its output, the system returned the reasoning conducted following to the mandated steps. In doing so, the result satisfied the formal validation criteria of evaluation as described in Section 3.1. For space reasons, in Figure 7 we show the last two steps:⁴⁵

```

1 [...]
2 3. Discarding functions that cannot be called in state @Q4:
3   - Discarded f2 because it can only be invoked in state @Q2.
4 4. Functions remaining after discarding those not callable in state @Q4, along with their pieces of code:
5
6 - f3:
7   ```
8 @Q4 Borrower: f3() {a} (a == 550) {
9   a -o Lender
10  a -> field4;
11
12 } ==> @Q5
13   ```
14
15 - fn (first occurrence):
16   ```
17 @Q4 Lender: fn() [] (field3 > 0) {
18   "notice" -> Borrower;
19   now + 48 >> @Q4 {"default" -> Borrower} ==> @Qd
20 } ==> @Q4

```

Figure 7. Final two steps (deciphering computable contracts)

⁴⁵ The full results can be consulted in the repo at <https://github.com/copmAI/copm/blob/main/src/stipula.ipynb>.

After identifying the specific states in which the functions can be called (Step 2), the system filtered out those that are not callable from @Q4 (lines 2–3). It then returned only the functions that can be invoked in state @Q4, namely f3 and fn.

This result is *correct* because the returned functions are indeed callable in @Q4, and at the same time *complete* because it includes all and only the relevant functions, providing exactly the needed information without omitting anything essential or adding unnecessary details. Thus, the answer fully meets the final two evaluation criteria.

4.1.2 Functions Explanation

The objective of this step is to obtain a detailed explanation of the functioning of the code extracted through the previous task. We encountered two main obstacles during our experiment. The first was a tendency by the LLM to overlook specific sections of the code, either skipping parts entirely – such as the condition for calling a function indicated in brackets – or simply offering only broad overall explanations of entire functions. Continuing, the second was the propensity to already interpret the code within a contractual context even if not asked. These kinds of assumptions at this stage clearly lack any reliability, since the LLM was only provided with a part of the contract code. Addressing the extraction of legal meaning from the code will be the focus of the next and final step in the pipeline, where the natural language contract is fed to the system to provide the necessary context.

We addressed the obstacles just described with the prompt shown in Figure 8.

```

1 You will receive a list of functions in the stipula programming language that are only callable from state @Q4.
2 Describe their runtime functioning according to the following structure.
3 - function: name the function - print the relevant piece of code
4 - state of invocation: name the state of invocation - print the relevant piece of code
5 - pre-conditions for invocation: describe the preconditions - print the relevant piece of code
6 - operations triggered: describe each operation and event described in the body of the function - print the relevant
  piece of code
7
8 Do not speculate on the legal meaning of the code.

```

Figure 8: Prompt 2 (deciphering computable contracts)

In an effort to address the first problem, we defined a precise structure for the answer, specifying each element the explanation should focus on and report. These refer to the main components of a function (lines 3, 4, 5) and to the operations defined in its body, such as field updates, asset transfers or scheduling of *events* (line 6). For each component, we specifically requested the system to show the specific piece of code analysed in order to be able to use this information for the next task. Finally, to overcome the second problem and avoid the tendency of freely ascribe legal meaning to the code, the prompt states a clear prohibition in this sense in line 8. Figure 9 shows the part of the answer focusing on one of the identified functions.

```

1 ### Function: fn (first occurrence)
2 - **Code:**
3 ...
4 @Q4 Lender: fn() [] (field3 > 0){
5     "notice" -> Borrower;
6     now + 48 >> @Q4 {"default" -> Borrower} ==> @Qd
7 } ==> @Q4
8 ...
9 - **State of Invocation:** @Q4
10 - **Pre-conditions for Invocation:** The value of 'field3' must be greater than 0.
11 - **Operations Triggered:**
12   - A "notice" is sent to the Borrower ('"notice" -> Borrower').
13   - An event is scheduled to occur 48 time units from the current time. If the contract is still in state @Q4 when
     this time expires, a "default" message is sent to the Borrower, and the contract transitions to state @Qd ('
     now + 48 >> @Q4 {"default" -> Borrower} ==> @Qd').
14   - The contract remains in state @Q4 after the function execution ('==> @Q4').

```

Figure 9. The part of the answer focusing on one of the identified functions (deciphering computable contracts)

As we can see, the provided answer is adherent to the prescribed structure, meeting the *formal validation* criterion. The response strictly adheres to the task of providing a technical explanation of the functions' behavior, avoiding any interpretation of legal meaning, as instructed. In doing so, it accurately explains the code's functionality. For instance, regarding the event in line 13, the system correctly captures its behaviour and properly identifies the role of the states involved – an aspect it had previously misinterpreted. Thus, we can conclude that the outcome is *correct*. Furthermore, every piece of code is included and explained, including the preconditions that were previously overlooked (line 28). As a result, the analysis is both complete and fully aligned with the required content, effectively meeting the *substantial validation* criteria.

4.1.3 Mapping Contract Code to Legal Provisions

Having extracted the right meaning of the contract code, the goal of this final task is to enable the user to understand how this relates to the contract text. In order for such an explanation to be clear and effective, we believe it must include four necessary elements:

1. the relevant piece of code
2. its technical meaning (obtained in the previous task)
3. the exact contract provision it refers to, if any, and
4. an explanation of how the code logic is implementing that provision.

To achieve this, we fed the natural language contract to the model together with the code explanations obtained in Section 4.1.2. We did not employ additional memory for this final task. This allowed the system to better focus on the essential data inherited from previous results. Throughout the experiment we encountered several issues. We immediately noticed that when presented with multiple functions, the system tended to summarize them altogether or to overlook relevant pieces of code. Most importantly, the LLM often assumes connections of pieces of code to contract provisions even when they do not have one. Due to the nature of computable contracts, in fact, some code elements may be necessary for the program logic but lacking a direct match with the legal natural language text.

By fine-tuning our approach, we achieved the following prompt, providing the system with one function at a time (Figure 10)

```

1 You have received the natural language contract and a the explanation of a function automating part of its lifecycle.
2 Identify, step by step, what each aspect of the function represent in the natural language contract.
3 Follow the structure:
4   contract: article title related to the function;
5   function: name of the function;
6   function aspect: describe one function operation representing/implementing the contract;
7   explanation: explain the match between contract and the aspect;
8   contract text: "citation of the specific contract text relating to the aspect"
9
10 Some aspects of the function can have no direct correspondence with the contract.
11 Indicate where there is no specific correspondence.
```

Figure 10. Prompt 3 (deciphering computable contracts)

The prompt provides a clear structure to carry out the analysis. The exact elements that have to be analysed are specified: the contract article to which the function refers (line 4), the specific contract text (line 8), the function's name (line 5) and the 'explanation' section (line 7), where the system has to explain the link between contract and code. By indicating 'function aspects' (line 6), we ensure that each code element extracted from the function body in the previous step (such as preconditions, operations, events, etc.) is covered and not overlooked by the system. Finally, to avoid legal interpretations of merely technical code elements, we prompted the system to indicate in the answer when a specific correspondence with the text could not be found, ensuring transparency in the analysis process (lines 10–11).

The answer shown in Figure 11 adheres to the provided structure, fulfilling the *formal criteria* similarly to previous tasks.

```

1 ### Function: fn (first occurrence)
2 - **contract: Events of Default**
3 - **function: fn()**
4 - **function aspect: now + 48 >> @Q4 {"default" -> Borrower} ==> @Qd**
5 - **explanation:** This operation schedules a transition to a default state (@Qd) if the Borrower does not cure the
   default within 48 time units from the notice, mirroring the contract's allowance of a two-day period for
   the Borrower to remedy the default after receiving notice. The use of "48 time units" in the function
   corresponds to the "two days" period specified in the contract for curing a default.
6 - **contract text:** "provided they shall remain uncured within a period of two days after notice is given to
   Borrower by Lender of their occurrence"
7 - **function aspect: Pre-conditions for Invocation: The value of 'field3' must be greater than 0.**
8 - **explanation:** This precondition likely represents a condition in the contract that must be met for a default
   notice to be issued, such as a missed payment or another obligation not being fulfilled by the Borrower.
   However, the contract does not explicitly mention 'field3', indicating this is a representation of a
   contractual condition in the function's logic rather than a direct citation.
9 - **contract text:** No specific correspondence.
```

Figure 11. Formal criteria (deciphering computable contracts)

The system effectively identifies and explains every piece of code in the input, avoiding the oversights of previous attempts and consistently associating each element with the corresponding contract text and article. As a result, the *substantial validation* criterion is satisfied.

The explanation in line 5 demonstrates the system's ability to establish a connection between the code logic and the relevant contract provision. It correctly recognises that the transition to the default state @Qd is triggered by the borrower's non-performance and that the '48 time units' specified in the code align with the two-day grace period stated in the contract.

The system accurately identifies cases where a direct match between the code and the contract text is absent, such as the condition 'field3 must be greater than 0' in line 7. While it initially provides a reasonable (and ultimately correct) explanation for this condition, it promptly clarifies that no explicit connection to the contract exists, ensuring transparency in the analysis process (line 8). However, we observed a slight tendency of the model to introduce inaccuracies when explaining pieces of code related to state transitions (e.g. line 10 of Listing 1). While it correctly captures their technical behaviour, it occasionally attributed legal meanings beyond what was strictly present in the code, interpreting purely technical transitions as carrying legal significance. For these reasons, we conclude that the *correctness* criterion is fulfilled to a great extent, though with minor inaccuracies.

4.2 Expert Systems – CrossJustice

The CrossJustice platform is a rule-based system capable of automatic legal reasoning in the domain of criminal law, providing users with meaningful information about their rights and obligations as suspects, or accused, of having committed a crime in the European Union (EU) member state's territory⁴⁶. CrossJustice was identified as suitable since it holds all the main characteristics we ought to look for in order to ground our approach. It provides a trace of the logical steps needed to reach an output; it uses the high-level Prolog programming language to reason over the applicable rules; its knowledge base is made up of several legal sources, each connected to different categories of rights; and, finally, it applies EU law as well as member state law to justify its inferences. In particular, the limitations of CrossJustice follow the standard constraints of rule-based systems, as also stated in Section 1. The rule-base contains a Prolog representation of the main articles from several EU Directives⁴⁷ and their national implementation in selected member states.⁴⁸

To introduce the factual scenario provided in Listing 6, let us imagine that a person, named Mario is involved in criminal proceedings taking place in the Netherlands (line 10), and does not speak Dutch (line 13). He has been presented with a document during the proceeding, and according to Article 3, paragraph 3, such a document has been deemed by the authority to be essential (line 12). Thus, according to Article 3, paragraph 1, of the European Directive 2010/64, Mario has the right to receive a translation of this document, which is essential to ensure that he is able to exercise his right of defence and to safeguard the fairness of the proceedings. This is the main right presented in the CrossJustice platform in lines 8–9, followed by a recap of the specific conditions needed for that right to be granted.

Moreover, in the CrossJustice system, we also express the relation between a primary right and the additional rights that expand the meaning and the implementation of the main one.

In this case, the defendant, Mario, has the right to have the costs of the translator covered by the state under Article 4 (lines 15–23); moreover, according to Article 3, paragraph 7, an oral translation of essential documents may be provided instead of a written translation under the condition that such oral translation does not prejudice the fairness of the proceedings (lines 25–34).

We will now explore the outcome of the same factual scenario, as applied in the Dutch legal system (Figures 12 and 13).

⁴⁶ Billi, "Explainability through Argumentation."

⁴⁷ Directives 2010/64, 2012/13, 2013/48, 2016/1919, 2016/343, 2016/800.

⁴⁸ Bulgaria, Netherlands, Italy, Poland, Spain.

```

1 directive_2010_64 - art3_1
2
3 Article 3
4 Option: essentialDocument
5
6 Explanation:
7
8 has_right(right_to_translation, dir, art3_1, mario, essentialDocument)
9   has_right(art3_1, mario, right_to_translation, essentialDocument)
10     proceeding_language(mario, dutch) [FACT]
11     essential_document(art3_3, mario, documents)
12     authority_decision(mario, essential_document) [FACT]
13     not(person_understands(mario, dutch))
14
15 Auxiliaries:
16
17 art4 - cost - state
18
19 Article 4
20 Explanation:
21
22 auxiliary_right(art4, art3_1, mario, cost, state)
23   auxiliary_right(art4, mario, cost, state)
24
25 Properties:
26
27 art3_7 - form - oral
28
29 Article 3.7
30 Explanation:
31
32 right_property(art3_7, art3_1, mario, form, oral)
33   right_property(art3_7, mario, form, oral)
34   not(proceeding_event(mario, prejudice_fairness))

```

Figure 12: Right to translation – EU Directive

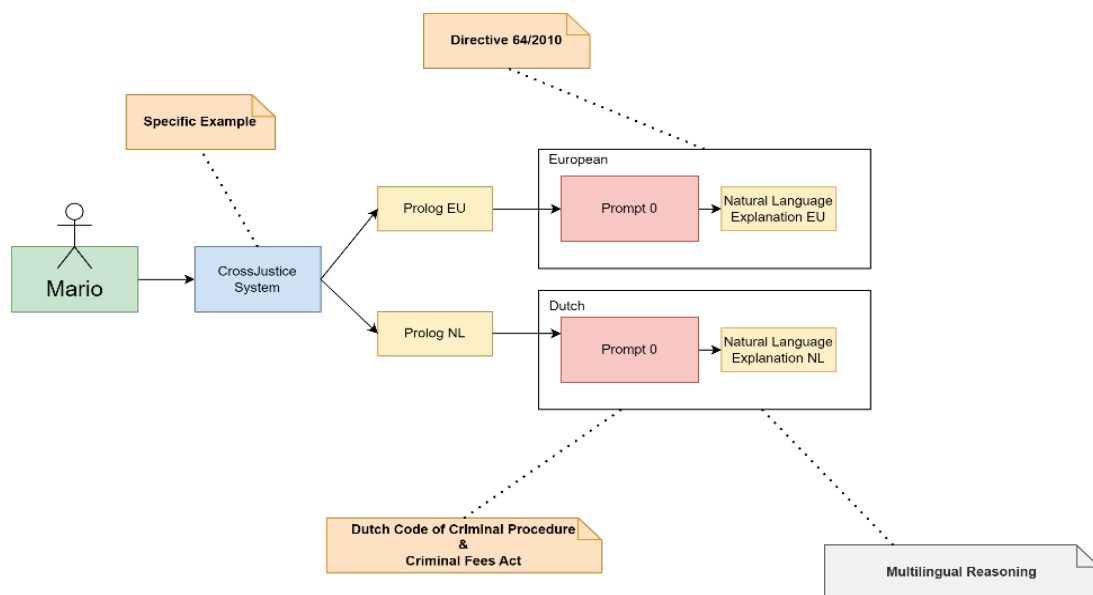


Figure 13. CoPM – natural language explanation

In this case, the person subject to the proceedings (Mario) is granted the same right under Article 32, paragraph a(1) of the Dutch code of Criminal Procedure (lines 8–9). The conditions for the right to be granted are different, as it is necessary for the person to submit a reasoned request to the authority to ask for a document to be recognised as essential (line 11), and therefore for the translation to be granted. To be clear, Article 3(3) of the Directive also sets out a suspect’s right to submit, at any stage of the criminal proceedings (within the meaning of Article 1 of the Directive), a ‘reasoned request’ to obtain the translation of documents, but this is not a mandatory requisite, and the judge can always act ‘ex officio’.

The stance taken by the legislator has sparked some criticism. In fact, the Dutch Bar Association (Nederlandse Orde van Advocaten) has called the current implementation of Article 3 ‘minimalistic’, thereby suggesting that implementing legislation should include a general obligation to obtain a translation of all documents on which it can rely to reach a finding of guilt.

Furthermore, this request needs to be submitted in writing and needs to specify which documents (or the parts thereof) are deemed ‘necessary’, with an indication of the reasons underpinning this request. Pursuant to Article 3(3) of the Directive, the competent authority decides on a case-by-case basis and may reject the request submitted to this effect by the suspect⁴⁹.

The right to have the costs of the translation met by the state is explicitly provided under Article 1, paragraph 4, of the Criminal Fees Act (*Wet tarieven in strafzaken*), but there is no applicable article regarding the form of the translation.

Our goal was for the model to be able to extract from the Prolog trace the following pieces of information:

- a simplified summary of the norms relevant to the inference
- the list of rights granted to the user according to the input facts representing the case
- a description of the inference process that led the system to its solution.

With these objectives in mind, we directed our efforts towards achieving the most flexible and versatile prompt, able to be applied to any of the inferences from the CrossJustice system (Figure 14).

```

1 directive_2010_64_nl - art32a_1
2
3 Article 32a(1) Dutch Code of Criminal Procedure
4 Option: essentialDocument
5
6 Explanation:
7
8 has_right(right_to_translation, nl, art32a_1, mario, essentialDocument)
9   has_right(art32a_1, mario, right_to_translation, essentialDocument)
10     person_status(mario, suspect) [FACT]
11     person_request_submitted(mario, essential_document) [FACT]
12
13 Auxiliaries:
14
15 art1_4 - cost - state
16
17 Article 1(4) The Criminal Cases Fees Act
18 Explanation:
19
20 auxiliary_right(art1_4, art32a_1, mario, cost, state)
21   auxiliary_right(art1_4, mario, cost, state)

```

Figure 14. Right to translation – Dutch law

Early experiments, without the addition of memories to the pipeline, resulted in the following issues. The first problem was obtaining a fixed structure in the output, which was often presented in different formatting (e.g. bullet lists, numbered lists, free-form text). We found that it was necessary to give a fixed structure for the LLM to follow, in order to decrease the degree of variance between answers and maximise repeatability and reliability. Second, we required the LLM to take into consideration all legal terms and facts from the scenario, and to apply those to build the explanation, as it had a propensity to miss key facts if not well prompted. For example, it would often miss the fact that the form of the translation could be oral, missing the argument and stating that it could be provided in a written form.

Finally, the most challenging aspect was to overcome the tendency of the model to ascribe meaning to legal text where not explicitly mentioned or provided. In particular, the meaning was often guessed based on the most general significance, disregarding the specificity of juridical norms and legal lexicon.

Generally, the model struggled to include all pertinent Prolog facts necessary for the application of legal reasoning to a given scenario, which falls under the criteria of *substantial validation*. Moreover, at first, it exhibited shortcomings in accurately capturing and depicting particular legal inferences derived from the Prolog trace, which involve the application of subordinate rules. With specific reference to our case, it would struggle in identifying the connection between lines 11 and 12 of Figure 12.

⁴⁹ Contissa, “Effective Protection.”

The study further observed the model’s challenges in precisely conveying the meaning of Prolog terms. An instance of this was noted when it misinterpreted a predicate concerning the qualification of an essential document. Instead of recognising that a document falls under the definition of ‘essential’, it would often change the meaning of the word to be more or less general. At this point of the research efforts, to solve the aforementioned issues we introduced two legal sources in the declarative memory, to have the model refer to the original legal articles in order to use the appropriate lexicon for the translation.

We started from the assumption that having the original text of the legal articles from which the Prolog is derived, the model would more efficiently use the correct terminology and provide more consistent answers. However, during the course of our experiments into the natural language translation of programming languages, we encountered several challenges related to semantic extraction, multilingual representation and data segmentation of the contextual information. First, we grappled with the issue of deriving semantic meaning from large blocks of memory, which proved to be overly complex and confusing. Our solution was to implement smaller, more manageable segments that allowed for a clearer semantic interpretation. This segmentation allowed for a more structured and focused approach to translation, ensuring that the semantic integrity of each segment was maintained.

Second, the integration of multilingual content posed a significant challenge. Initially, we attempted to combine two languages within the same memory context, but this approach was dominated by English, overshadowing the other language and leading to an imbalance. It became evident that our embedding model tended to perceive texts in the same language, or particularly in English, as more similar, regardless of their actual content. To address this, we opted to separate the languages, dedicating distinct memory spaces to each, which improved the quality and relevance of the translation.

Therefore, we divided Directive 2010/64 into 12 segments and the Dutch Code of Criminal Procedure into around 100 segments, each segment being an article of the legal source. This way, we could ensure that each had only one topic for the model to focus on, and evaluate the relevance to the prompt.

The CoPM approach allowed the system to chain together relevant legal rules based on the specific facts of a case. Furthermore, the LLM was able to easily parse the Dutch text and extract the relevant norms, while still providing a response in English. In the final prompt engineered for this task, we first introduce the object and the form of the input data. Second, we provide a fixed structure to follow (Summary What Rights do You Have – Why do You Have Them). This is aimed at guaranteeing that the model followed the given formal criteria in order to maximise repeatability and reliability. Furthermore, we specifically requested the system to use all the Prolog terms in the explanation, with explicit references to the original. This was aimed at providing an easier way to read and evaluate the answer, both from a substantial and juridical point of view. For a complete overview of input, prompts, outputs, and contextual data, refer to the online repository.

As can be seen in Figure 15, the result is presented in an accessible and readable way, upholding the instructions given about structure and formatting (*formal validation*).

```

1 Summary:
2 [...]right to have essential documents translated [...]
3
4 What Rights do You Have:
5 1. Right to Translation of Essential Documents: [...]
6 2. Right to Challenge and Complain: [...]
7
8 Why do You Have Them:
9 1. Right to Translation of Essential Documents: This right is based on the fact that you do not understand the
   language of the proceedings (not(person_understands(mario, dutch))) and that there are documents considered
   essential for your defense (essential_document(art3_3, mario, documents)) as determined by the authority (
   authority_decision(mario, essential_document)).
10
11 2. Right to Challenge and Complain: This is derived from the procedural safeguards outlined in the directive,
   ensuring that you can seek redress if you believe the translation services provided are inadequate or
   unnecessary.
12 [...]

```

Figure 15. The result, presented in an accessible and readable way

It also included all relevant Prolog facts used to apply juridical reasoning to the specific case (*substantial validation*), and it grasped and correctly represented all the legal inferences of CrossJustice.

Continuing, we can see how the relevant articles have been correctly identified from memories, reflecting the system’s ability to extract relevant information from the legal source and apply it to the specific case. This is clear from line 11, where it shows new information (e.g. the right to challenge and complain about the quality of the translation) which is not present in the Prolog source code but is granted by Article 3, paragraph 5, of the same Directive.

The same prompt was additionally applied to the Dutch legal source, and all three criteria have again been fully satisfied. The system was able to provide a good explanation of the case study, correctly identifying all relevant facts and their meaning as applied to the user. It also identified the relevant articles in the Dutch Code of Criminal Procedure, but did not provide any additional information that was not already present in the Prolog code, although the right to challenge a decision is also present in the Dutch legislation.

5. Conclusions

This work introduced the CoPM approach, a hybrid approach that integrates the justifiable and trustworthy features of symbolic models with the natural language processing capabilities of LLMs. By enhancing accessibility and versatility, this methodology provides a structured framework for the interaction and interpretation of symbolic systems by laypeople.

The CoPM methodology was applied to computable contracts, demonstrating its ability to extract technical meaning from contract-related code and establish clear connections between legal agreements and their computational representations. The implementation leveraged the Stipula programming language, a domain-specific procedural language designed for encoding legal contracts. Through a structured pipeline, including targeted code retrieval, technical analysis and natural language processing, the approach effectively linked contract provisions with their corresponding programmatic expressions.

Furthermore, CoPM was validated through its application to the CrossJustice platform, a rule-based system for automatic legal reasoning in the domain of criminal law. The methodology enabled the system to trace logical reasoning steps, extract relevant legal norms and generate structured outputs while maintaining linguistic adaptability. By integrating LLM capabilities, CoPM successfully facilitated the interpretation of Dutch legal texts while ensuring that responses were produced in English, demonstrating its potential for multilingual legal applications. The expert evaluation confirms the effectiveness of CoPM in augmenting symbolic legal systems and strengthening their reasoning capabilities.

Through this hybrid approach, this work represents a step towards bridging the gap between symbolic reasoning and natural language processing to create more transparent, explainable and effective legal automation systems. Future research will focus on testing the methodology on different symbolic legal systems, further validating the soundness of the hybrid approach and assessing its applicability across distinct representations of legal knowledge. Additionally, efforts will be directed toward scaling and refining the CoPM architecture, with promising indications of its capacity to construct complex and modular frameworks for legal reasoning.

Bibliography

- Allen, Jason Grant. “Wrapped and Stacked: ‘Smart Contracts’ and the Interaction of Natural and Formal Language.” In *Smart Legal Contracts: Computable Law in Theory and Practice*, edited by Jason Allen and Peter Hunn, 23–53. Oxford: Oxford University Press, 2022.
- Armitage, Matthew. “Trust, Confidence, and Automation: The ISDA Master Agreement as a Smart Contract.” *Business Law Review* 43 (2022): 56–64. <https://doi.org/10.54648/BULA2022009>.
- Barrera, Meritxell Fernandez and Pompeu Casanovas. “From User Needs to Expert Knowledge: Mapping Laymen Queries with Ontologies in the Domain of Consumer Mediation.” In *AI Approaches to the Complexity of Legal Systems. Models and Ethical Challenges for Legal Systems, Legal Language and Legal Ontologies, Argumentation and Software Agents – International Workshop AICOL-III, Held as Part of the 25th IVR Congress, Frankfurt am Main, Germany, August 15–16, 2011*, edited by Monica Palmirani, Ugo Pagallo, Pompeu Casanovas and Giovanni Sartor, 286–308. Cham: Springer, 2011. https://doi.org/10.1007/978-3-642-35731-2_20.
- Billi, Marco, Roberta Calegari, Giuseppe Contissa, Giuseppe Pisano, Galileo Sartor and Giovanni Sartor. “Explainability Through Argumentation in Logic Programming.” In *Proceedings of the International Conference on Logic Programming 2021 Workshops Co-located with the 37th International Conference on Logic Programming (ICLP 2021), Porto, Portugal (Virtual)*. 2021. <https://ceur-ws.org/Vol-2970/causalpaper2.pdf>.
- Billi, Marco, Alessandro Parenti, Giuseppe Pisano and Marco Sanchi. “A Hybrid Approach for Accessible Rule-based Reasoning through Large Language Models.” *arXiv*, Preprint Version. 2024. <https://doi.org/10.48550/arXiv.2311.11811>.
- Billi, Marco, Alessandro Parenti, Giuseppe Pisano and Marco Sanchi. “ALEXChat – a Generative Symbolic Approach to Legal Explainability.” In *Proceedings of the Twentieth International Conference on Artificial Intelligence and Law (ICAIL ’25)*. 2025.
- Billi, Marco, Giuseppe Pisano and Marco Sanchi. “Fighting the Knowledge Representation Bottleneck with Large Language Models.” In *Legal Knowledge and Information Systems – JURIX 2024: The Thirty-seventh Annual Conference, Brno, Czech Republic, 11-13 December 2024*, edited by Jaromir Savelka, Jakub Harasta, Tereza Novotna, and Jakub Misek, 14–24. Amsterdam: IOS Press, 2024. <https://doi.org/10.3233/FAIA241230>.
- Blair-Stanek, Andrew, Nils Holzenberger and Benjamin Van Durme. “Can GPT-3 Perform Statutory Reasoning?” In *Proceedings of the Nineteenth International Conference on Artificial Intelligence and Law, ICAIL 2023, Braga, Portugal, June 19–23, 2023*, edited by Matthias Grabmair, Francisco Andrade, and Paulo Novais, 22–31. ACM, 2023. <https://doi.org/10.1145/3594536.3595163>.
- Chang, Yupeng, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen et al. “A Survey on Evaluation of Large Language Models.” *ACM Transactions on Intelligent Systems and Technology* 15, no 3 (2024): 39:1–39:45. <https://doi.org/10.1145/3641289>.
- Charrow, Robert P. and Veda R. Charrow. “Making Legal Language Understandable: A Psycholinguistic Study of Jury Instructions.” *Columbia Law Review* 79, no 7 (1979): 1306–1374. <https://www.jstor.org/stable/pdf/1121842.pdf>.
- Chien, Colleen V. and Miriam Kim. “Generative AI and Legal Aid: Results from a Field Study and 100 Use Cases to Bridge the Access to Justice Gap.” *Loyola of Los Angeles Law Review*, 2024. <https://ssrn.com/abstract=4733061>.
- Chien, Colleen V., Miriam Kim, Akhil Raj and Rohit Rathish. “How Generative AI Can Help Address the Access to Justice Gap Through the Courts.” *Loyola of Los Angeles Law Review*, 2024. <https://ssrn.com/abstract=4683309>.
- Contissa, Giuseppe, Giulia Lasagni, Michele Caianiello and Giovanni Sartor. *Effective Protection of the Rights of the Accused in the EU Directives: A Computable Approach to Criminal Procedure Law*. Leiden: Brill, 2022.
- Crafa, Silvia, Cosimo Laneve, Giovanni Sartor and Adele Veschetti. “Pacta Sunt Servanda: Legal Contracts in *Stipula*.” *Science of Computer Programming* 225 (2023): 102911. <https://doi.org/10.1016/J.SCICO.2022.102911>.
- Deroy, Aniket, Kripabandhu Ghosh, and Saptarshi Ghosh. “Ensemble Methods for Improving Extractive Summarization of Legal Case Judgements.” *Artificial Intelligence Law* 32, no 1 (2024): 231–89. <https://doi.org/10.1007/S10506-023-09349-8>.
- Doshi-Velez, Finale, Mason Kortz, Ryan Budish, Chris Bavitz, Sam Gershman, David O’Brien, Stuart Schieber, James Waldo, David Weinberger and Alexandra Wood. “Accountability of AI Under the Law: The Role of Explanation.” *CoRR* abs/1711.01134 (2017). <http://arxiv.org/abs/1711.01134>.
- Flood, Mark D. and Oliver R. Goodenough. “Contract as Automaton: Representing a Simple Financial Agreement in Computational Form.” *Artificial Intelligence Law* 30, no 3 (2022): 391–416. <https://doi.org/10.1007/S10506-021-09300-9>.
- Gao, Yunfan, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai et al. “Retrieval-Augmented Generation for Large Language Models: A Survey.” *CoRR* abs/2312.10997 (2023). <https://doi.org/10.48550/ARXIV.2312.10997>.
- Garimella, Aparna, Abhilasha Sancheti, Vinay Aggarwal, Ananya Ganesh, Niyati Chhaya and Nandakishore Kambhatla. “Text Simplification for Legal Domain: Insights and Challenges.” In *Proceedings of the Natural Legal Language Processing Workshop, NLLP@EMNLP 2022, Abu Dhabi, United Arab Emirates (Hybrid), December 8, 2022*, edited by

- Nikolaos Aletras, Ilias Chalkidis, Leslie Barrett, Catalina Goanta, and Daniel Preotiuc-Pietro, 296–304. Association for Computational Linguistics, 2022. <https://doi.org/10.18653/V1/2022.NLLP-1.28>.
- Governatori, Guido, Trevor J. M. Bench-Capon, Bart Verheij, Michal Araszkiewicz, Enrico Francesconi and Matthias Grabmair. “Thirty Years of Artificial Intelligence and Law: The First Decade.” *Artificial Intelligence Law* 30, no 4 (2022): 481–519. <https://doi.org/10.1007/S10506-022-09329-4>.
- Janatian, Samyar, Hannes Westermann, Jinzhe Tan, Jaromir Savelka and Karim Benyekhlef. “From Text to Structure: Using Large Language Models to Support the Development of Legal Expert Systems.” In *Legal Knowledge and Information Systems — JURIX 2023: The Thirty-sixth Annual Conference, Maastricht, The Netherlands, 18–20 December 2023*, edited by Giovanni Sileno, Jerry Spanakis and Gijs van Dijck, 167–76. Amsterdam: IOS Press, 2023. <https://doi.org/10.3233/FAIA230962>.
- Katz, Daniel Martin, Michael James Bommarito, Shang Gao, and Pablo Arredondo. “GPT-4 Passes the Bar Exam.” *Philosophical Transactions of the Royal Society A* 382, no. 2270 (2024): 20230254. <https://doi.org/10.1098/rsta.2023.0254>.
- Kojima, Takeshi, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo and Yusuke Iwasawa. “Large Language Models are Zero-Shot Reasoners.” In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28–December 9, 2022*, edited by Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh. 2022. <http://papers.nips.cc/paper%5Cfiles/paper/2022/hash/8bb0d291acd4acf06ef112099c16f326-Abstract-Conference.html>.
- Kowalski, Robert A., Jacinto A. Davila, Galileo Sartor and Miguel Calejo. “Logical English for Law and Education.” In *Prolog: The Next 50 Years*, edited by David Scott Warren, Veronica Dahl, Thomas Eiter, Manuel V. Hermenegildo, Robert A. Kowalski and Francesca Rossi, 287–99. Cham: Springer, 2023. https://doi.org/10.1007/978-3-031-35254-6_24.
- Laneve, Cosimo, Alessandro Parenti and Giovanni Sartor. “Draft Better Contracts.” In *Legal Knowledge and Information Systems – JURIX 2024: The Thirty-seventh Annual Conference, Brno, Czech Republic, 11-13 December 2024*, edited by Jaromir Savelka, Jakub Harasta, Tereza Novotna; and Jakub Misek, 95–106. Amsterdam: IOS Press, 2024. <https://doi.org/10.3233/FAIA241237>.
- Laneve, Cosimo, Alessandro Parenti and Giovanni Sartor. “Legal Contracts Amending with Stipula.” In *Coordination Models and Languages - 25th IFIP WG 6.1 International Conference, COORDINATION 2023, Held as Part of the 18th International Federated Conference on Distributed Computing Techniques, DisCoTec 2023, Lisbon, Portugal, June 19–23, 2023, Proceedings*, edited by Sung-Shik Jongmans and Antonia Lopes, 253–270. Cham: Springer, 2023. https://doi.org/10.1007/978-3-031-35361-1_14.
- Liang, Percy, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang et al. “Holistic Evaluation of Language Models.” *Transactions on Machine Learning Research* (2023). <https://openreview.net/forum?id=iO4LZibEqW>.
- Ligeza, Antoni. *Logical Foundations for Rule-based Systems*. Cham: Springer, 2006.
- Lymer, Andrew and Ken Richards. “A Hybrid-Based Expert System for Personal Pension Planning in the UK.” *Intelligent Systems in Accounting, Finance and Management* 4, no 1 (1995): 71–88. <https://doi.org/10.1002/j.1099-1174.1995.tb00080.x>.
- Martineau, Kim. “What is Retrieval-Augmented Generation?” *IBM Blog*, 2023.
- Masri, Naser, Yousef Abu Sultan, Alaa N. Akkila, Abdelbaset Almasri, Adel Ahmed, Ahmed Y Mahmoud, Ihab Zaqout and Samy S Abu-Naser. “Survey of Rule-based Systems.” *International Journal of Academic Information Systems Research (IJAISR)* 3, no 7 (2019): 1–23. <http://ijeais.org/wp-content/uploads/2019/07/IJAISR190701.pdf>.
- Masson, Michael E. J. and Mary Anne Waldron. “Comprehension of Legal Contracts by Non-experts: Effectiveness of Plain Language Redrafting.” *Applied Cognitive Psychology* 8, no 1 (1994): 67–85. <https://doi.org/10.1177/0739456X18774122>.
- Mellinkoff, David. *The Language of the Law*. Boston: Little, Brown & Co, 1963.
- Merigoux, Denis, Nicolas Chataing and Jonathan Protzenko. “Catala: A Programming Language for the Law.” *Proceedings of ACM Programming Language* 5, no. ICFP (2021): 1–29. <https://doi.org/10.1145/3473582>.
- Michaelsen, Robert H. “An Expert System for Federal Tax Planning.” *Expert Systems* 1, no 2 (1984): 149–67.
- Mittelstadt, Brent D., Chris Russell and Sandra Wachter. “Explaining Explanations in AI.” In *Proceedings of the Conference on Fairness, Accountability, and Transparency, FAT* 2019, Atlanta, GA, USA, January 29–31, 2019*, edited by danah boyd and Jamie H. Morgenstern, 279–288. ACM, 2019. <https://doi.org/10.1145/3287560.3287574>.
- Morris, Jason. “Blawx: Rules as Code Demonstration.” *MIT Computational Law Report*, August 2020. <https://law.mit.edu/pub/blawxrulesascodedemonstration>.
- Nguyen, Ha Thanh, Randy Goebel, Francesca Toni, Kostas Stathis and Ken Satoh. “LawGiBa – Combining GPT, Knowledge Bases, and Logic Programming in a Legal Assistance System.” In *Legal Knowledge and Information Systems – JURIX 2023: The Thirty-sixth Annual Conference, Maastricht, The Netherlands, 18–20 December 2023*, edited by Giovanni Sileno, Jerry Spanakis, and Gijs van Dijck, 371–74. Amsterdam: IOS Press, 2023. <https://doi.org/10.3233/FAIA230991>.

- Nguyen, Ha Thanh, Francesca Toni, Kostas Stathis and Ken Satoh. "Beyond Logic Programming for Legal Reasoning." In *Proceedings of the International Conference on Logic Programming 2023 Workshops co-located with the 39th International Conference on Logic Programming (ICLP 2023), London, July 9th and 10th, 2023*, edited by Joaquin Arias, Sotiris Batsakis, Wolfgang Faber, Gopal Gupta, Francesco Pacenza, Emmanuel Papadakis, Livio Robaldo et al., vol. 3437. CEUR Workshop Proceedings. CEUR-WS.org, 2023. https://ceur-ws.org/Vol-3437/paper_2LPLR.pdf.
- Paquin, Louis-Claude, Francois Blanchard and Claude Thomasset. "Loge-Expert: From a Legal Expert System to an Information System for Non-Lawyers." In *Proceedings of the Third International Conference on Artificial Intelligence and Law, ICAIL '91, Oxford, England, June 25–28, 1991*, edited by Richard E. Susskind, 254–59. ACM, 1991. <https://doi.org/10.1145/112646.112678>.
- Sanchi, Marco and Tereza Novotna. "Towards Hybrid Evaluation Methodologies for Large Language Models in the Legal Domain." In *Legal Knowledge and Information Systems – JURIX 2024: The Thirty-seventh Annual Conference, Brno, Czech Republic, 11–13 December 2024*, edited by Jaromir Savelka, Jakub Harasta, Tereza Novotna, and Jakub Misek, 389–92. Amsterdam: IOS Press, 2024. <https://doi.org/10.3233/FAIA241279>.
- Savelka, Jaromir, Kevin D. Ashley, Morgan A. Gray, Hannes Westermann and Huihui Xu. "Explaining Legal Concepts with Augmented Large Language Models (GPT-4)." *CoRR* abs/2306.09525 (2023). <https://doi.org/10.48550/ARXIV.2306.09525>.
- Schauer, Frederick. *Playing by the Rules: A Philosophical Examination of Rule-based Decision-making in Law and in Life*. Oxford: Clarendon Press, 1991.
- Sottana, Andrea, Bin Liang, Kai Zou and Zheng Yuan. "Evaluation Metrics in the Era of GPT-4: Reliably Evaluating Large Language Models on Sequence to Sequence Tasks." In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6–10, 2023*, edited by Houda Bouamor, Juan Pino and Kalika Bali, 8776–8788. Association for Computational Linguistics, 2023. <https://doi.org/10.18653/V1/2023.EMNLP-MAIN.543>.
- Sun, Zhongxiang. "A Short Survey of Viewing Large Language Models in Legal Aspect." *CoRR* abs/2303.09136 (2023). <https://doi.org/10.48550/ARXIV.2303.09136>.
- Surden, Harry. "Computable Contracts." *UCDL Review* 46 (2012): 629.
- Susskind, Richard. *Expert Systems in Law: A Jurisprudential Inquiry*. Oxford: Oxford University Press, 1987.
- Trautmann, Dietrich. "Large Language Model Prompt Chaining for Long Legal Document Classification." *CoRR* abs/2308.04138 (2023). <https://doi.org/10.48550/ARXIV.2308.04138>.
- Trautmann, Dietrich, Alina Petrova and Frank Schilder. "Legal Prompt Engineering for Multilingual Legal Judgement Prediction." *CoRR* abs/2212.02199 (2022). <https://doi.org/10.48550/ARXIV.2212.02199>.
- Uijttenbroek, Elisabeth M., Arno R. Lodder, Michel C. A. Klein, Gwen R. Wildeboer, Wouter Van Steenbergen, Rory L. L. Sie, Paul E. M. Huygen, and Frank van Harmelen. "Retrieval of Case Law to Provide Layman with Information about Liability: Preliminary Results of the BEST-Project." In *Computable Models of the Law, Languages, Dialogues, Games, Ontologies*, edited by Pompeu Casanovas, Giovanni Sartor, Nuria Casellas and Rossella Rubino, 291–311. Cham: Springer, 2008. https://doi.org/10.1007/978-3-540-85569-9_19.
- van Melle, William. "MYCIN: A Knowledge-based Consultation Program for Infectious Disease Diagnosis." *International Journal of Man–Machine Studies* 10, no 3 (1978): 313–22. [https://doi.org/10.1016/S0020-7373\(78\)80049-2](https://doi.org/10.1016/S0020-7373(78)80049-2).
- Wei, Jason, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le and Denny Zhou. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models." In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28–December 9, 2022*, edited by Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh. 2022. <http://papers.nips.cc/paper%5Cfiles/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html>.
- Westermann, Hannes, Sébastien Meeùs, Mia Godet, Aurore Clément Troussel, Jinzhe Tan, Jaromir Savelka and Karim Benyekhlef. "Bridging the Gap: Mapping Layperson Narratives to Legal Issues with Language Models." In *Proceedings of the 6th Workshop on Automated Semantic Analysis of Information in Legal Text co-located with the 19th International Conference on Artificial Intelligence and Law (ICAIL 2023), Braga, Portugal, 23rd September 2023*, 37–48. 2023. <https://ceur-ws.org/Vol-3441/paper5.pdf>.
- Zin, May Myo, Ken Satoh, and Georg Borges. "Leveraging LLM for Identification and Extraction of Normative Statements." In *Legal Knowledge and Information Systems – JURIX 2024: The Thirty-seventh Annual Conference, Brno, Czech Republic, 11–13 December 2024*, edited by Jaromir Savelka, Jakub Harasta, Tereza Novotna, and Jakub Misek, 215–225. Amsterdam: IOS Press, 2024. <https://doi.org/10.3233/FAIA241247>.