Consistent Formalization of Legal Text via Large Language Models

Shalini Panthangi

CMU-CS-25-146

December 2025

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Umut Acar, Chair
Sherry Tongshuang Wu

Submitted in partial fulfillment of the requirements for the Masters degree in Computer Science.

Keywords: Legal Knowledge Representation, Automated Legal Reasoning, Defeasible Deontic Logic, Neurosymbolic Systems, Large Language Models, Symbol Grounding, Formal Rule Extraction

For my parents.

Abstract

Precise logical formalization of legal text helps automated compliance analysis and machine-readable legal reasoning, which help streamline and prove complex queries in industries like law and insurance. Achieving this is challenging, as legal text includes ambiguity, exceptions, and layered nuances that make it difficult to consistently translate into logical rules. Existing large language model-based methods often generate inconsistent predicates, drift in meaning, and fail to capture complex legal structures. This thesis introduces a structured pipeline for converting legal text into Defeasible Deontic Logic and First-Order logic with a focus on keeping key terms consistent and grounding predicates in a stable manner. The approach introduces consistency through a symbol-table framework that constrains LLM outputs to a vocabulary of legal actors and actions. Combined with clause segmentation, multi-stage LLM rewriting, and automated Z3 consistency verification, the system produces logical rules that better maintain the intended argument structure of legal statutes. Evaluating with multiple legal examples shows that this method reduces logical errors and produces formalizations suitable for reasoning tasks and analysis. The results demonstrate that integrating symbolic guidance with LLM-based processing provides a path toward generating trustworthy formal representations of legal text.

Acknowledgments

I am deeply grateful to my advisor, Professor Umut Acar, for his mentorship and support. His patience and encouragement were vital to the completion of this thesis.

I also want to thank Professor Sherry Wu for her time and valuable insights as a member of my committee.

I am especially thankful to Mark Stehlik, whose early support opened the door for me to join this program.

To Mike Rainey: thank you for being such a great teammate and mentor. I've learned so much from our collaboration and truly appreciated your enthusiasm and inspiring work ethic.

To my family, thank you for everything. Your love and belief in me made everything possible for me, and I am so lucky to have had you in my corner through every challenge.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

As legal systems become increasingly complex and interconnected with digital infrastructure, there is a growing need for computational tools that can interpret, analyze, and reason about statutory requirements. Automated compliance systems, model-based legal analysis, and machine-readable regulatory frameworks all rely on transforming natural-language legal text into logical representations. However, statutory language is written for human interpretation, not machine consumption. It contains ambiguity, nested obligations, exceptions, priorities, and cross-referenced conditions that make it difficult to directly translate this text into structured logical forms.

Recent progress in large language models (LLMs) has made it possible to extract structured information from unstructured text, but these models struggle to maintain consistency across multi-step legal reasoning tasks. Direct LLM-to-logic approaches often introduce semantic drift, inconsistent predicate structures, and incomplete representations of obligations or exceptions. These issues make the resulting logical rules difficult to use for automated reasoning, and they prevent the reliability needed for high-stakes legal applications. This motivates the need for a method that brings together the flexibility of LLMs with the stability and precision of symbolic formalisms.

1.2 First-Order Logic

First-Order Logic (FOL) is a formal symbolic system used to represent and reason about objects, their properties, and the relations between them. Unlike propositional logic, which deals with atomic statements, FOL provides the expressive power necessary to quantify over individuals and define general rules that apply across a domain. In the context of legal formalization, FOL serves as the foundational layer for representing the structure of legal rules, allowing for the precise encoding of actors, actions, and the conditional relationships that define a legal framework.The syntax of FOL is built upon a vocabulary of constants, variables, functions, and predicates. A legal statement is represented as a formula where predicates define the characteristics of individuals and the relationships between them. For example, a requirement for an actor to hold a

specific license can be expressed as:

$$\forall x(\text{is\_actor}(x) \land \text{performs\_action}(x, a) \rightarrow \text{holds\_license}(x, l))$$

where x is a variable representing an individual, and the universal quantifier $\forall$ indicates that the rule applies to all entities satisfying the antecedent.A critical advantage of FOL is its use of *quantification* and *n-ary predicates*, which allow for the representation of complex legal dependencies that involve multiple participants or environmental conditions. In this work, FOL is utilized to provide the structural backbone for symbol grounding. By mapping natural language phrases to specific predicates and arguments, the pipeline ensures that the resulting logical forms maintain a consistent internal structure. This structural rigor is a prerequisite for automated theorem proving and model checking, as it allows reasoning engines to unify variables across different rules and verify whether a specific set of facts satisfies the conditions of a legal statute. While it lacks the native defeasibility of specialized logics, it provides the semantics required for core predicate grounding and relational mapping. By translating legal text into FOL-based representations, the system creates a high-fidelity map of the legal domain that can be further augmented with deontic modalities or defeasible priorities in subsequent reasoning stages.

1.3 Defeasible Deontic Logic

Defeasible Deontic Logic (DDL) is a formal logical framework designed to represent and reason about normative systems involving obligations, permissions, prohibitions, and exceptions. Unlike classical deontic logic, which treats norms as absolute, DDL explicitly supports *defeasible* reasoning, allowing rules to be overridden by more specific or higher-priority rules when conflicts arise. This property makes DDL particularly well-suited for modeling legal and regulatory texts, where general provisions are frequently qualified by exceptions, special cases, or contextual constraints.

At a high level, DDL represents norms as rules that associate a *deontic* modality with an action under certain conditions. A typical rule takes the form

$$\text{Antecedent} \Rightarrow [M]\ \text{Action},$$

where the antecedent is a (possibly empty) conjunction of conditions, M is a deontic modality, and the action represents the regulated behavior. The primary modalities considered in this work are *permission*, *prohibition*, and *obligation*.

A key feature of DDL is its ability to model legal exceptions through *rule priorities*. When two rules apply in the same situation but have conflicting modalities, DDL resolves the conflict by selecting the rule with higher priority. For example, a general permission may be overridden by a more specific prohibition that applies under exceptional circumstances. These override relationships are explicitly encoded using priority relations of the form

$$overrides(r_i, r_j),$$

indicating that rule r_i supersedes rule r_j when both are applicable.

This mechanism reflects a fundamental principle of legal reasoning: more specific or stronger norms take precedence over more general ones. For instance, a statute may grant permission to perform an action in general, while later prohibiting that same action for a particular class of actors or during certain conditions. DDL captures this structure without making the logic to become inconsistent.

Another important aspect of DDL is its separation of rule applicability from rule strength. A rule may be applicable in a given context if its antecedent holds, but it may still be defeated by a higher-priority rule. This distinction allows DDL to represent layered legal reasoning, where multiple norms coexist but only some ultimately determine the legal outcome in a specific scenario.

In the context of this thesis, DDL serves as an intermediate representation. It allows the system to encode permissions, prohibitions, and obligations extracted from legal text, while also modeling exceptions and override relationships in a principled way. By translating segmented and symbol-grounded rules into DDL, the pipeline preserves the structure of legal argumentation and prepares the rules for downstream reasoning and verification using symbolic tools.

1.4 Problem Statement and Goal

The main challenge of this work is to convert natural-language statutory text into a set of logical rules that can be reliably used by automated reasoning systems. Achieving this requires more than extracting surface-level structure. It requires internal consistency across the entire rule set.

Consistency is critical because even small variations, such as different predicate names for the same concept, mismatched argument structures, or inconsistent treatment of actors and actions, can cause automated reasoning tools to misinterpret rules or produce contradictory outcomes. In legal contexts, these inconsistencies undermine the ability to evaluate obligations, detect violations, or reason about exceptions with any degree of correctness. However, statutory text contains ambiguity, exceptions, and cross-references that make consistent translation difficult. And while large language models (LLMs) can capture rich semantic patterns, they often introduce synonym drift, inconsistent predicate generation, and structural variability, especially across multi-step frameworks.

The goal of this thesis is to develop a pipeline that formalizes legal text into Defeasible Deontic Logic (DDL) and first-order logic (FOL) while enforcing consistency in predicate names, argument structure, modality representations, and rule interactions. This approach aims to preserve statutory meaning while producing output stable enough for downstream reasoning tools such as Z3 and DDL engines, ultimately enabling more reliable automated analysis of legal text.

1.5 Contributions

This thesis contributes a structured and consistency-driven approach to formalizing legal text using a hybrid of rule-based, symbolic methods and large language models. First, it introduces a symbol-table–guided framework that constrains LLM outputs to a controlled vocabulary of legal actors, actions, and predicate schemas, to maintain stability across rules.

The work also introduces a multi-stage LLM agent that transforms raw statutory language into normalized text, structured clause representations, defeasible deontic logic, and ultimately, an attempt at generating first-order logic, with each stage designed to preserve meaning and enforce coherent argument structures. The system incorporates clause segmentation and predicate grounding techniques that extract actors, actions, conditions, and exceptions from legal text and map them into consistent logical forms. To ensure the correctness of the resulting rule sets, the work integrates a consistency verification system, allowing the produced logical formulas to be tested. This work demonstrates the feasibility of symbol-guided LLM formalization and lays groundwork for more reliable automated legal reasoning.

This work does not aim to determine legal correctness or authoritative interpretation. Instead, it focuses on representational reliability: whether the same legal text can be translated into stable, structurally consistent logical rules suitable for automated reasoning. Correctness remains a downstream concern for legal experts and formal validation frameworks.

Chapter 2

Related Works

This work builds on prior research in automated legal formalization, spanning early natural language processing (NLP) approaches to rule extraction, more recent large language model (LLM)-based methods for legal reasoning, and neurosymbolic systems that combine natural language understanding with formal verification. While each of these research directions contributes important insights, they also reveal limitations that motivate the consistency-driven, symbol-guided pipeline proposed in this thesis.

2.1 NLP-Based Rule Extraction from Legal Text

Before the emergence of large language models, several approaches attempted to extract legal rules using traditional NLP techniques. A representative example is *Combining Natural Language Processing Approaches for Rule Extraction from Legal Documents* [2], which proposes a pipeline based on syntactic parsing and rule-based pattern matching to identify normative statements in legal text.

This work relies on a Combinatory Categorial Grammar (CCG) parser to analyze sentence structure and extract deontic components such as obligations, permissions, and prohibitions. The system identifies candidate sentences, extracts relevant terms, annotates them with deontic modalities, and generates formal rules accordingly. An example mapping presented in the paper is:

$$\text{Complaint} \Rightarrow [O]\ \text{inform\_proposed\_resolution\_15\_days}$$

Evaluation is performed by comparing the automatically generated rule set against a gold-standard rule set manually produced by a human analyst. Metrics include the number of correctly extracted normative sentences, the number of correct terms identified, the accuracy of deontic component annotation, the number of correct rules generated from the extracted terms, and the impact of the CCG parser in supporting the discovery of new rule-generation patterns.

While this approach demonstrates that syntactic structure can be leveraged to extract legal rules, it depends heavily on manually designed patterns and brittle parsing assumptions. The resulting rules are limited in expressiveness and struggle to capture complex legal text, such as

nested exceptions, cross-referenced conditions, and priority relationships between rules. Moreover, the approach lacks a mechanism for grounding extracted terms in a stable vocabulary, making it difficult to maintain consistency across larger documents or multiple runs.

2.2 LLM-Based Legal Formalization into Defeasible Deontic Logic

More recent work explores the use of large language models to directly translate legal text into formal representations. *From Legal Texts to Defeasible Deontic Logic via LLMs: A Study in Automated Semantic Analysis* [3] investigates whether modern LLMs can generate Defeasible Deontic Logic (DDL) representations from statutory language.

In this work, multiple LLMs are prompted with legal text and tasked with producing DDL-style rules. The study examines the ability of different models to capture obligations, permissions, prohibitions, and exceptions, highlighting the flexibility of LLMs in handling complex legal language. Evaluation primarily focuses on qualitative analysis of the generated rules and comparison across different model architectures.

However, because the models generate rules independently, the resulting representations often exhibit inconsistency across runs. Common issues include variation in predicate naming, inconsistent argument structures, and instability in modality assignment. The absence of a grounding or constraint mechanism means that semantically equivalent concepts may be represented using different symbols, which limits the usability of the output for automated reasoning or verification. These limitations motivate the need for approaches that stabilize and constrain LLM outputs when used for legal formalization.

2.3 Neurosymbolic Formalization and Verification

A more recent research paper investigates neurosymbolic approaches that combine natural language interpretation with symbolic reasoning and formal verification. *A Neurosymbolic Approach to Natural Language Formalization and Verification* [1] presents a framework in which natural-language specifications are translated into formal logical representations that can be checked using SMT solvers.

In this approach, neural models are used to interpret and structure natural-language input, while symbolic reasoning tools are employed to verify logical properties of the resulting formalization. Evaluation focuses on whether the generated formulas satisfy or violate specified constraints, demonstrating that combining neural interpretation with formal verification can improve robustness over purely neural methods.

While this work illustrates the benefits of integrating neural and symbolic components, it does not explicitly address the problem of symbol consistency across rules or across multiple formalization runs. The mapping from language to logical predicates is largely unconstrained, which can result in semantic drift or incompatibility between generated formulas. As a result, verification outcomes may depend more on representational variability than on the underlying legal meaning.

2.4 Positioning of This Work

The approach proposed in this thesis draws inspiration from all three lines of research while addressing their core limitations. Like early NLP-based systems, it emphasizes structured extraction and explicit representation of deontic concepts, but it avoids brittle hand-crafted rules by leveraging LLMs. Like recent LLM-based approaches, it benefits from the expressive power of large language models, but it constrains them through a symbol-table–guided framework that enforces consistency across predicates, arguments, and modalities. Finally, like neurosymbolic systems, it integrates formal verification via Z3 and DDL reasoning engines, but it ensures that the inputs to these systems are grounded in a stable and coherent symbolic vocabulary.

By explicitly focusing on consistency and symbol grounding, this work bridges the gap between expressive language models and the strict requirements of formal legal reasoning, enabling the generation of logical rule sets that are both semantically faithful and suitable for automated analysis.

Chapter 3

Methodology

The methodology for this work is built around a multi-stage pipeline that turns natural-language legal text into a consistent and well-structured set of formal rules. Legal language is often dense, filled with exceptions, and highly dependent on context, so the pipeline separates the task into smaller components that mirror how legal meaning is naturally constructed. These components use a combination of rule-based and LLM-assisted techniques to identify important phrases, align them with a stable vocabulary, break the text into individual rules, convert those rules into logical forms, and check them for consistency. The following sections describe each part of this pipeline in more detail, showing how the text moves from atomic phrase extraction, to symbol table matching, to rule segmentation, to formal rule creation, and finally to logic encoding, as shown in Figure 3.1. A full example walkthrough of this methodology is in Chapter 4.

3.1 Legal Text Preprocessing

The first stage of the pipeline prepares the raw statutory input so it can be reliably analyzed, segmented, and formalized. Legal text often appears in long, compound sentences with overlapping obligations, coordinated actors, embedded conditions, and extraneous details that make direct parsing difficult. To address this, the preprocessing stage normalizes the text and restructures it into a clearer form before any symbolic or logical processing occurs.

3.1.1 Input Normalization

The pipeline begins by taking the original statutory paragraph and cleaning it into a format suitable for linguistic analysis. This includes removing noise such as unnecessary punctuation, flattening formatting inconsistencies, and ensuring the text is treated as a single unified input. The goal at this stage is not to interpret the meaning but to create a clean representation that downstream components can analyze reliably.

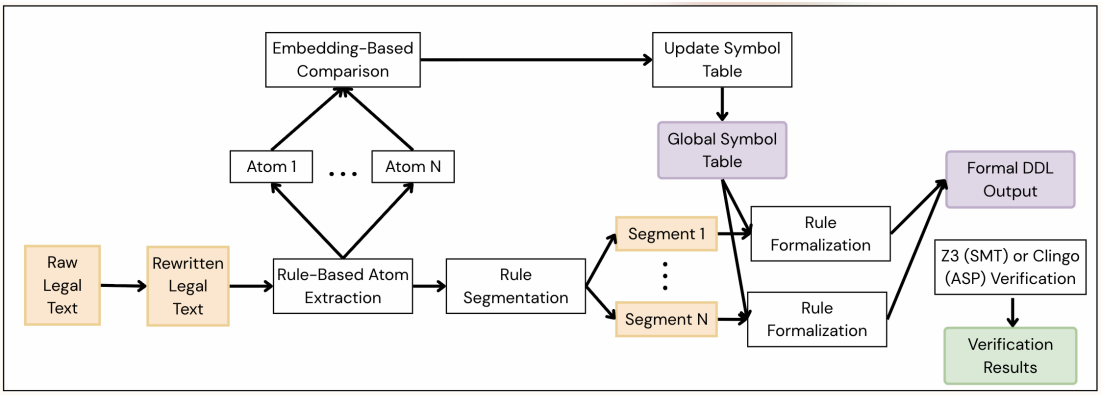

Figure 3.1: Overview of the proposed pipeline method for formalizing legal text into consistent logical rules.

3.1.2 LLM-Assisted Rule Rewriting

Because statutory sentences may combine multiple duties or permissions within a single clause, the system uses an LLM to rewrite the text into a clearer and more standardized set of sentences. This rewriting step performs several key transformations:

- Separates compound sentences into individual statements that each express one obligation, permission, or prohibition.
- Rephrases complex constructions into simpler, more direct forms while retaining the original meaning.
- Standardizes modality by expressing obligations, permissions, and prohibitions using consistent, explicit phrasing (e.g., ”may,” ”are prohibited from,” ”can”).
- Unpacks coordinated actors (e.g., ”law enforcement officers OR certified first responders”) into separate, parallel sentences.
- Preserves conditions and exceptions such as emergency scenarios or training requirements.

The result is a rewritten rule set where each sentence corresponds to one potential formal rule. This structured rewriting greatly reduces ambiguity and minimizes the risk of losing critical details later in the pipeline. The full prompt used for this step is provided in Appendix A.2.

3.2 Atomic Phrase Extraction

The second stage of the pipeline identifies the fundamental semantic units that appear in the rewritten legal text. These units include actors, actions, objects, and relevant modifiers, and they form the vocabulary used throughout the rest of the formalization process. Extracting them early in a rule-based manner ensures that every later transformation is grounded in the structure of the original statutory language, rather than arbitrarily introduced terms from the LLMs.

3.2.1 Dependency-Based Term Extraction

To extract atomic phrases, the system applies dependency parsing to the simplified legal sentences produced during preprocessing. Using the syntactic structure of each sentence, the pipeline identifies individual subject–verb–object configurations, capturing the core propositional content of the text.

The extraction process focuses on:

- Subjects (actors): noun phrases that perform or are affected by an action (e.g., Civilians, Licensed Contractors).
- Verbs (actions): the core legal behaviors expressed in the text (e.g., Operate, Hold, Complete).
- Objects and complements: the entities that actions operate upon or relate to (e.g., Municipal Drones, Safety Training).
- Prepositional attachments: additional structured information, such as "in an emergency" or "with an active permit," recorded as separate atomic extras.

The extraction procedure expands nominal phrases to include compound modifiers and "of"-phrases, ensuring important details (e.g., equipment safety training, active municipal permit) are preserved. Each extracted phrase corresponds to one potential predicate or argument in the later logic representation. Once the atomic phrases are identified, the system normalizes them into a consistent, machine-usable form. This canonicalization step standardizes lexical variation and ensures that semantically identical terms are treated as the same symbol throughout the pipeline.

Canonicalization includes:

- Cleaning and normalizing text: removing punctuation, articles, and other extraneous items.
- Applying consistent casing: converting terms into CamelCase (e.g., "certified first responders" $\rightarrow$ CertifiedFirstResponders).
- Ensuring stable predicate names: mapping verb lemmas and noun phrases to canonical forms so they can be reliably compared, matched, and reused.

The output of this stage is a set of unique canonical atoms that represent all semantically relevant entities and actions present in the input text. These atoms form the basis for symbol table matching, rule building, and the later generation of logical predicates.

3.3 Symbol Table Matching

After extracting atomic phrases from the text, the pipeline aligns these phrases with an existing controlled vocabulary stored in the symbol table. The symbol table serves as a consistency mechanism: it ensures that identical concepts across different rules, sections, or documents map to the same predicate name, preventing drift and fragmentation in the logical representation. Symbol table matching is therefore a crucial step in maintaining coherence across the entire formalization pipeline. Over time, as more legal text is processed, the symbol table becomes a richer and more stable representation of the legal domain, reducing redundancy and improving the accuracy of future matches.

3.3.1 Embedding-Based Similarity Scoring

For each canonicalized atom extracted from the text, the system computes a semantic similarity score between the atom and each entry in the current symbol table. This is done using vector embeddings generated by OpenAI's embedding API, which provide a representation of each atom's meaning in a high-dimensional semantic space.

Each atom is then paired with the closest symbol table entry based on cosine similarity. The purpose of this matching procedure is to detect when two different terms (e.g. LicensedContractors vs Contractors or Operate vs Operating) are referring to the same underlying concept. When a high-confidence match is found with a score above a certain threshold, the atom is treated as an instance of the existing symbol rather than a new one.

This step promotes semantic stability across the pipeline, ensuring that the system does not create redundant or inconsistent predicate symbols for similar legal concepts.

Not all extracted atoms will have a close match in the existing symbol table. When the similarity score does not meet the threshold for a reliable match, the pipeline classifies the atom as representing a new legal concept and inserts it into the symbol table. This dynamic update process allows the symbol table to expand as it encounters new actors or actions across different texts, building a richer and more complete vocabulary over time.

Each newly added symbol becomes available for future matching, allowing the pipeline to learn and stabilize terminology across multiple examples, sections, or documents. In practice, this mechanism allows the system to adapt to domain-specific terminology and statutory phrasing without losing the internal consistency required for formal reasoning.

3.3.2 Integration of Statutory Definitions

Most legal documents include a dedicated definitions section, where key terms are assigned explicit meanings with nuanced conditions, exceptions, etc. These definitions form the vocabulary of the statute, determining how obligations and rights apply throughout the document. To take advantage of this structure, the pipeline allows predefined terms from such definitions sections to be loaded into the symbol table before any analysis begins, following the usage described above.

Initializing the symbol table using statutory definitions accomplishes several important goals:

- It anchors the vocabulary to the law's own terminology.
- It reduces unnecessary symbol proliferation by giving the matching stage a stronger base to compare against.
- It promotes consistency across sections that rely on the same defined terms.

By incorporating definitions upfront, the pipeline uses the statute's own terminology, which helps prevent confusion between similar terms and ensures that any concept used in a formal rule can be easily traced back to an explicit definition in the law.

3.4 Rule Segmentation

After the symbol table has been updated, the pipeline turns to identifying the individual normative statements embedded in the rewritten legal text. Even after simplification, legal text

frequently bundles together distinct permissions, prohibitions, conditions, or actor-specific provisions that must be separated before they can be formalized. The rule segmentation stage isolates these components into discrete rule units and annotates each with preliminary modality and structural information needed for subsequent logical translation.

3.4.1 Identifying Distinct Rule Units

The rewritten text produced in the preprocessing stage is passed to an LLM that segments the content into individual rules. Each resulting rule captures a single legal proposition—typically one actor, one action, and one modality. This segmentation ensures that complex sentences such as those involving multiple groups (”law enforcement officers or certified first responders”) or multiple conditions (”who has completed training and holds a permit”) are unpacked into separate, clearly defined rule units.

Each segmented rule includes several key pieces of metadata:

- A natural-language rule statement that captures the cleaned, simplified form of the norm.
- A modality hint (e.g., ”may,” ”must not,” ”shall not”) extracted from the text to guide later formalization into ”permitted,” ”obligated,” and ”forbidden”.
- A reason-role label, identifying whether the rule is a main provision, an exception, or a stronger, overriding exception.
- An isolated clause structure, ensuring that each rule expresses exactly one relationship.

This structured intermediate representation provides a stable foundation for the construction of formal DDL and FOL rules. This segmentation is performed using a prompt that asks the model to identify modality, exception structure, and priority relationships between rules. The complete prompt for rule segmentation is included in Appendix A.3.

3.4.2 Exception and Priority Detection

Legal texts frequently contain exceptions, which are rules that apply only under special conditions or that override more general provisions. To capture these relationships, the pipeline includes a dedicated step that identifies when a segmented rule represents an exception, a stronger prohibition, or a context-specific limitation on another rule.

This detection is performed using a combination of linguistic cues and LLM interpretation. The system looks for markers which signal that a rule modifies or supersedes a preceding one. When such cues are present, the pipeline classifies the rule with an appropriate reason-role (e.g., MAIN, EXCEPTION, STRONG EXCEPTION).

For rules marked as exceptions or stronger prohibitions, the pipeline also determines which earlier rules they override. For example, if a statute first states that *”certain actors have permission to operate equipment during an emergency,”* but later states that *”authorized personnel flagged for misuse are prohibited from doing so, even in an emergency,”* the second rule must take precedence if an authorized personnel is tagged for misuse. The system encodes this by assigning priority relationships such as *”r5 overrides r3”* or *”r5 overrides r4”*.

These priority annotations make it possible for the later logic engine to correctly reason about conflicts between rules, determine which rules remain applicable in specific contexts, and enforce

the standard legal principle that exceptions preempt general permissions.

3.5 Rule Formalization

Once the text has been segmented into discrete rule units and annotated with modality hints, reason roles, and priority relationships, the next stage of the pipeline uses an LLM to convert these natural-language rules into structured logical representations. This formalization step synthesizes the grounded symbols, extracted conditions, and segmented rule statements into Defeasible Deontic Logic (DDL).

During this step, the LLM receives a rule segment produced in the previous step, including the rewritten natural-language version of the rule, the identified actor and action phrases, any accompanying conditions extracted from the text, and a modality hint indicating whether the rule expresses a permission, prohibition, or obligation. The LLM also receives access to the current symbol table, which contains the canonical vocabulary of actors, actions, and predicate names collected so far that are the most relevant to the current text. Together, these inputs give the model both the linguistic context of the rule and the semantic constraints needed to generate a consistent logical representation. The LLM uses this information to construct a precise predicate form, integrate the relevant conditions into the rule's antecedent, and map the rule's modality to the appropriate deontic category. Formal rule construction is guided by a highly constrained prompt that enforces strict symbol reuse, predicate arity limits, and syntactic validity for downstream logical solvers. This prompt is intentionally restrictive to prevent hallucinated entities, implicit variables, or inconsistent predicate naming. The full formalization prompt is included in Appendix A.4.

3.5.1 Rule Components and Structure of the Final Rules

Before describing the individual components of the pipeline, it is helpful to outline the structure of the formal rules produced by the system. Each rule is ultimately expressed in a uniform format that aligns with Defeasible Deontic Logic (DDL) and supports automated reasoning over permissions, prohibitions, obligations, and exceptions, with the ability to reason about rule overrides as well. Each formal rule generated by the pipeline contains the following core elements:

- **Head**: The principal action being regulated, represented as a predicate such as

  $$Operate(Civilians, MunicipalDrones).$$

  The head captures the legal behavior or event that the rule governs.

- **Modality**: The force assigned to the rule, derived from the text and mapped to one of the DDL operators:

  $$\text{PERMISSION},\ \text{FORBIDDEN},\ \text{OBLIGATION}.$$

  This determines whether the action is allowed, prohibited, or required.

- **Antecedent**: A (possibly empty) conjunction of conditions under which the rule applies. For example:

  $$LicensedContractors \wedge Complete(\dots) \wedge Hold(\dots) \Rightarrow \text{PERMISSION } Operate(\dots).$$

  If the rule has no conditions, the antecedent defaults to $\top$, indicating that the rule applies universally.
- **Priority Relations**: For rules that function as exceptions or stronger prohibitions, the system records override relationships of the form

  $$overrides(r_i, r_j),$$

  indicating that rule r_i supersedes rule r_j when both could apply. This mechanism enables defeasible reasoning and correctly models legal exceptions such as "however" or "even during an emergency."

3.5.2 Resulting Representation

The combination of these components yields a structured rule of the form

$$\text{Antecedent} \Rightarrow \text{[Modality] Head}$$

along with any associated override relationships. This format allows the system to capture the meaning of the statute in a precise, machine-readable way while supporting conflict resolution, exception handling, and downstream reasoning tasks.

3.5.3 Predicate Construction

For each segmented rule, the system constructs a predicate that represents the legal action described in the text. This involves combining the canonicalized actors, actions, and objects identified earlier into a structured logical form. For example, a statement such as "Civilians are not allowed to operate municipal drones" becomes a predicate of the form:

$$Operate(Civilians, MunicipalDrones)$$

The symbol table guides this construction by ensuring that subjects, verbs, and objects map to stable, previously grounded symbols. If the text expresses additional components. such as training requirements, permit status, or emergency contexts, these may appear as separate predicates connected in the antecedent of the rule. The result is a consistent predicate representation that captures both the action and its participants.

3.5.4 Modality Mapping

Each rule segment includes a modality hint derived from the rewritten text (e.g., "may," "shall not," "must not"). During formalization, these cues are mapped onto the corresponding DDL operators:

- **PERMISSION** for rules indicating that an action is allowed,
- **FORBIDDEN** for prohibitions or negative permissions, and
- **OBLIGATION** for mandatory actions (when present).

This mapping ensures that the formalized rule expresses not only what action is being described but also the normative force associated with it. The explicit use of deontic operators allows downstream reasoning engines to evaluate permissions, prohibitions, and obligations in a logically consistent manner.

3.5.5 Condition Integration

Many legal rules apply only under specific conditions, such as during a declared emergency, after completing safety training, or when holding a valid permit. The formalization stage integrates these conditions by placing them in the antecedent of the rule. For example:

$$LicensedContractors \wedge Complete(\dots) \wedge Hold(\dots) \Rightarrow \text{PERMISSION } Operate(\dots)$$

Similarly, exception clauses and stronger prohibitions identified earlier (e.g., "even during an emergency") are represented as conditions that interact with the priority structures established in the rule segmentation stage.

3.6 Logic Encoding (Meta-Theory Construction)

Once the rule formalization stage produces a set of structured Defeasible Deontic Logic (DDL) rules, the final step of the pipeline encodes these rules into executable logical systems that support automated reasoning and consistency checking. This work targets two reasoning backends: an SMT-based encoding using Z3, and a rule/ASP-based encoding using Clingo. Together, these encodings allow for both satisfiability checking and defeasible reasoning over permissions, prohibitions, and exceptions.

3.6.1 Encoding Rules for SMT Solving with Z3

Z3 is a state-of-the-art Satisfiability Modulo Theories (SMT) solver developed by Microsoft Research that is widely used for formal verification, program analysis, and automated reasoning. In principle, Z3 would be an ideal backend for this work, as it supports rich first-order logic, implication, and model generation, which align well with the structure of formalized legal rules.

In practice, however, encoding defeasible reasoning and rule override semantics in Z3 proved difficult within the scope of this project. While Z3 excels at checking logical consistency and producing explanations for why particular conclusions hold or fail, it does not natively support priority-based rule defeat, which is central to legal exception handling. As a result, Z3 was primarily explored as a potential solver, but practically was not able to fully be implemented.

3.6.2 Encoding Defeasible Rules in Clingo

Clingo is an answer set programming (ASP) system that supports non-monotonic reasoning through stable model semantics, making it particularly effective for representing defeasible rules, exceptions, and priority relationships common in legal texts. Unlike classical logical systems that require global consistency, Clingo naturally accommodates rule overriding and contextual applicability, allowing more specific or higher-priority norms to defeat more general ones. In this work, Clingo enables direct execution of Defeasible Deontic Logic rules by encoding modalities, conditions, and override relations within a declarative rule-based framework. Its ability to compute multiple answer sets provides transparent insight into how legal conclusions are derived, making it a powerful tool for modeling exception-driven legal reasoning and exploring alternative normative outcomes under different contexts.

To support defeasible reasoning and explicit exception handling, the pipeline also encodes the formalized rules into a logic program suitable for execution by the Clingo answer set solver. Clingo provides native support for non-monotonic reasoning, making it well-suited for modeling the priority-based semantics of Defeasible Deontic Logic.

In this encoding, each rule is represented as a fact that specifies its identifier, modality, and head predicate. Conditions are represented as separate facts associated with each rule, and override relationships are encoded explicitly. A simplified representation takes the form:

```
rule(r1, forbidden, operate(civilians, municipal_drones)).
condition(r1, emergency).
overrides(r2, r1).
```

A meta-theory written in Clingo defines when a rule is applicable, when it is defeated by a higher-priority rule, and which deontic conclusions ultimately hold. Applicability is determined by whether all conditions associated with a rule are satisfied. A rule is defeated if there exists another applicable rule with higher priority that prescribes a conflicting modality for the same action.

This encoding allows Clingo to compute stable models that reflect legally valid outcomes, resolving conflicts between general rules and their exceptions. Queries such as whether an action is permitted or prohibited in a given context can be answered directly by examining the resulting answer sets.

3.6.3 Scenario Evaluation and Condition Assignment

To evaluate a concrete scenario, the system is given a potential scenario in text and applies a very similar pipeline as described above to turn the scenario text into possible facts. This helps determine which condition predicates are true or false based solely on the scenario description and the rules generated by the law text related to that scenario. This step is performed using a separate prompt that asks the language model to interpret the scenario and assign truth values to each condition appearing in the Clingo rules, without inferring unstated facts or legal conclusions. The full prompt for condition evaluation is included in Appendix A.5.

Given a scenario, the resulting condition assignments are injected into the Clingo program as facts such as:

```
holds(emergency).
not holds(has_valid_permit).
```

Clingo is then executed on the combined rule set, condition facts, and meta-theory to compute stable models representing legally valid outcomes. Queries such as whether a particular action is permitted or prohibited in the given scenario are answered by inspecting the resulting answer sets.

Chapter 4

Case Study Examples and Outputs

Overview. This case study illustrates how the proposed pipeline formalizes a statutory provision containing a general obligation with a conditional exception. Such exception-driven structures are common in insurance and regulatory law and pose challenges for direct language-to-logic approaches due to conflicting modalities and priority relationships.

Input Legal Text. The input consists of the following statutory fragment:

> *An insurer must provide coverage for emergency medical treatment. However, the insurer is not required to provide coverage for policyholders that have committed insurance fraud.*

This text expresses a general obligation followed by an exception that restricts its applicability.

Rewrite and Normalization. In the first stage of the pipeline, the input text is rewritten by an LLM into a normalized form that simplifies sentence structure and resolves discourse markers such as "however." The rewritten text is:

> *An insurer must provide coverage for emergency medical treatment. If the policyholder has committed insurance fraud, then the insurer is not required to provide coverage for emergency medical treatment.*

This step isolates conditional structure explicitly and produces a representation that is easier to parse and segment. The rewrite is performed deterministically to reduce variability across runs.

Atomic Phrase Extraction. Next, the normalized text is processed using rule-based dependency parsing to extract atomic subject–verb–object structures. From the rewritten text, the following core triples are identified:

Subject	Verb	Object
Insurer	Provide	Coverage
Policyholder	Commit	InsuranceFraud

These atomic components form the candidate actors, actions, and objects used throughout the remainder of the pipeline.

Symbol Table Update. Each extracted atom is compared against the existing symbol table using embedding-based similarity. In this example, none of the extracted atoms exceed the similarity threshold required to reuse an existing symbol. As a result, the symbols `Insurer`, `Provide`, `Coverage`, `Policyholder`, and `InsuranceFraud` are added to the symbol table as new canonical entries. This step ensures that subsequent rules referring to these concepts use a stable and consistent vocabulary.

Rule Segmentation and Priority Detection. The rewritten text is then segmented into individual rule units. The pipeline identifies two distinct rules:

> r_1: *"An insurer must provide coverage for emergency medical treatment," labeled as a* MAIN *rule with an obligation modality.*
> r_2: *"If the policyholder has committed insurance fraud, the insurer is not required to provide coverage for emergency medical treatment," labeled as an* EXCEPTION *with a prohibition modality.*

The pipeline further determines that r_2 has higher priority than r_1, reflecting the exception structure in the original text.

Rule Formalization. Finally, each rule segment is converted into an intermediate Defeasible Deontic Logic representation. The resulting formal rules are:

$$r_1 : \top \Rightarrow [O]\ \mathit{Provide}(\mathit{Insurer}, \mathit{Coverage}, \mathit{EmergencyMedicalTreatment})$$
$$r_2 : \ \mathit{Commit}(\mathit{Policyholder}, \mathit{InsuranceFraud}) \Rightarrow$$
$$[F]\mathit{Provide}(\mathit{Insurer}, \mathit{Coverage}, \mathit{EmergencyMedicalTreatment})$$

with an explicit priority relation indicating that r_2 overrides r_1.

Final Clingo Program

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FACTUAL RULE ENCODING
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% r1: Default obligation
rule(r1, obligation, provide(insurer, coverage,
```

```
emergencymedicaltreatment)).

% r2: Exception | prohibition when fraud is committed
rule(r2, forbidden, provide(insurer, coverage,
emergencymedicaltreatment)).
condition(r2, commit(policyholder, insurancefraud)).

% r2 overrides r1 (exception defeats default obligation)
overrides(r2, r1).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% META-THEORY (DEFEASIBLE DEONTIC LOGIC)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

\% A rule is applicable if all its conditions hold
applicable(R) :-
    rule(R, _, _),
    not not_applicable(R).

not_applicable(R) :-
    condition(R, C),
    not holds(C).

\% A rule is defeated if a higher-priority conflicting rule
applies defeated(R) :-
    overrides(R2, R),
    applicable(R2),
    conflicts(R, R2).

\% Conflict definitions (same action, opposing modalities)
conflicts(R1, R2) :-
    rule(R1, obligation, A),
    rule(R2, forbidden, A).

conflicts(R1, R2) :-
    rule(R1, forbidden, A),
    rule(R2, obligation, A).

conflicts(R1, R2) :-
    rule(R1, permission, A),
    rule(R2, forbidden, A).

conflicts(R1, R2) :-
```

```
    rule(R1, forbidden, A),
    rule(R2, permission, A).

\% A rule holds if it is applicable and not defeated
holds_rule(R) :-
    applicable(R),
    not defeated(R).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% DERIVED NORMATIVE CONCLUSIONS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

obligation(A) :-
    holds_rule(R),
    rule(R, obligation, A).

forbidden(A) :-
    holds_rule(R),
    rule(R, forbidden, A).

permission(A) :-
    holds_rule(R),
    rule(R, permission, A).
```

Discussion. This example demonstrates how the proposed pipeline preserves the structure of legal reasoning by separating general obligations from their exceptions and encoding priority relationships explicitly. By grounding predicates in a shared symbol table and producing a defeasible representation, the resulting rules are suitable for downstream reasoning engines such as Clingo and Z3. In contrast to direct LLM-based formalization, this approach yields stable predicates, explicit override relationships, and a representation that closely mirrors the logical intent of the original statute.

Chapter 5

Results

5.1 Baseline: Direct LLM-to-Logic Formalization

As a baseline, we evaluate a zero-shot approach in which a large language model is prompted to directly translate natural-language policy statements into logical rules, without any intermediate rewriting, symbol grounding, or structural constraints. Each query is provided to the model independently, and the model is asked to produce a logic-style representation of the rule using its own inferred predicate names, argument structure, and condition encoding. No shared vocabulary, symbol table, or consistency enforcement mechanism is provided across queries.

Table 5.1 presents representative outputs produced by this baseline for two semantically related queries. The examples shown are illustrative rather than exhaustive and are intended to highlight common qualitative failure modes observed in practice. In particular, while both queries describe permissions and prohibitions over the same underlying action and actor class, the generated rules differ in predicate arity, variable usage, and condition structure. These discrepancies are not isolated artifacts of the specific examples shown, but are representative of broader inconsistencies observed across repeated runs and across related inputs, as quantified in the consistency metrics reported in table 5.2.

5.2 Experimental Setup and Data Generation

To evaluate the stability and consistency of rule generation, we conducted controlled repeated-run experiments for both the zero-shot baseline and the proposed pipeline. The goal of this setup was not to measure task accuracy against a gold standard, but rather to quantify how much the generated formalizations change when the same method is applied multiple times to the same input text.

5.2.1 Input Examples

The evaluation was performed on a set of ten legal-style examples designed to reflect common patterns in legal text, including role-based permissions, conditional access, prohibitions, obliga-

Query	Natural Language Input	Baseline LLM Output

Query 1	The user can edit the network settings if they are an IT staff member.	`rule(q1r1, permission, edit_network_settings).`
		
		`condition(q1r1, is_it_staff_member).`

Query 2	IT staff members without a permit cannot edit the network settings.	`rule(q2r1, forbidden, edit_network_settings(Actor)) :- it_staff_member(Actor), not permit(Actor).`

Table 5.1: Baseline LLM-generated formalizations for two semantically related queries. Despite referring to the same underlying action and actor class, the outputs exhibit inconsistent predicate structure, argument usage, and condition encoding, making joint reasoning over the rules difficult.

tions, and explicit exceptions. Each example consists of a short natural-language policy statement and does not rely on external context or prior examples.

5.2.2 Independent Repeated Runs

For each example, both methods were executed independently three times:

- The **zero-shot baseline** consists of directly prompting a large language model to generate logic rules from the input text without any intermediate rewriting, symbol grounding, or structural constraints.
- The **proposed pipeline** applies a multi-stage transformation process, including clause segmentation, symbol-table grounding, guided rewriting, and structured rule construction, before producing the final logical representation.

Each run was performed independently, with no shared state, caching, or reuse of intermediate outputs between runs. This design ensures that any observed variation reflects stochasticity in the generation process rather than artifacts of implementation or execution order.

5.2.3 Collected Outputs

For each run, the system outputs a set of ASP-style rules encoding modalities, predicate heads, and associated conditions. In the case of the pipeline, additional structured metadata (e.g., formal rule objects and priority relations) is produced but was not used directly in computing the consistency metrics in order to maintain parity with the zero-shot baseline.

All rule outputs were serialized to a common textual format and subsequently parsed into a normalized internal representation capturing:

- the rule modality (permission, prohibition, obligation),

- the predicate head, and
- the set of antecedent conditions.

5.2.4 Metric Computation

The Rule Consistency Score and Modality Stability metrics were computed by comparing the three runs for each example within the same method. Importantly, comparisons were performed *within* a method rather than across methods. This isolates the effect of structural constraints on generation stability and avoids conflating consistency with expressiveness or modeling choices.

Averaged scores were then reported across all ten examples to provide a high-level comparison of stability characteristics between the zero-shot baseline and the proposed pipeline.

5.2.5 Scope and Limitations

Because the evaluation focuses on repeated-run consistency rather than semantic correctness, these metrics do not assess whether a generated rule is legally correct or complete. Instead, they measure properties that are critical for symbolic reasoning systems: reproducibility, structural coherence, and robustness to stochastic variation. As such, the results should be interpreted as evidence of improved reliability rather than definitive correctness.

5.3 Consistency and Stability Metrics

To evaluate the internal consistency of generated rule sets and the stability of deontic interpretations across repeated runs, we introduce two automated metrics: *Rule Consistency Score* (RCS) and *Modality Stability*. These metrics are designed to capture properties that are critical for downstream logical reasoning but are often overlooked by surface-level accuracy measures.

5.3.1 Rule Consistency Score (RCS)

The Rule Consistency Score measures how structurally consistent the generated rules are across multiple runs of the same method on identical input text. For each example, we generate three independent outputs and compare all pairwise combinations. Each rule is parsed into a normalized representation consisting of its modality, head predicate, and set of conditions. Two rules are considered identical only if all three components match exactly.

Formally, for a given example with three runs, the RCS is defined as the average of the pairwise structural similarity scores, where similarity is binary (1 for an exact match, 0 otherwise), normalized by the maximum number of rules produced in either run. An RCS of 1.0 indicates perfect agreement across runs, while lower scores indicate divergence in rule structure, predicate naming, or condition attachment.

This metric directly captures a key failure mode of LLM-based formalization: producing different logical structures for the same legal text across repeated invocations.

Method	Avg. Rule Consistency	Avg. Modality Stability
Zero-shot LLM	0.03	0.36
Proposed Pipeline	0.61	0.42

Table 5.2: Average consistency and stability metrics across ten examples. Higher values indicate greater agreement across repeated runs.

5.3.2 Modality Stability

Modality Stability measures whether the deontic classification of a rule (e.g., PERMISSION, FORBIDDEN, OBLIGATION) remains consistent across runs for the same underlying action. For each canonical predicate head (e.g., `enter`, `operate`, `approve`), we collect the modalities assigned to that predicate across all runs. A predicate is considered stable if the same modality is assigned in every run.

The Modality Stability score for an example is the fraction of predicate heads whose modalities are fully consistent across runs. This metric reflects whether a system reliably preserves the normative force of legal statements, independent of syntactic variation. In legal reasoning contexts, inconsistent modality assignment can lead to contradictory conclusions even when predicates appear similar.

5.3.3 Quantitative Results

Table 5.2 summarizes the average Rule Consistency Score and Modality Stability for both the proposed pipeline and a zero-shot LLM baseline across ten test cases.

5.3.4 Interpretation

The results show a substantial improvement in Rule Consistency when using the proposed symbol-guided pipeline. The zero-shot baseline exhibits near-zero consistency, indicating that repeated runs frequently differ in predicate structure, argument placement, or condition encoding. This variability makes the resulting rule sets unsuitable for reliable symbolic reasoning, as even small structural differences can change solver behavior.

In contrast, the pipeline achieves a markedly higher Rule Consistency Score, demonstrating that constraining predicate construction through a symbol table and staged rewriting significantly reduces structural drift. While consistency is not perfect—reflecting remaining sources of ambiguity in condition handling and exception modeling—the improvement highlights the effectiveness of explicit grounding and controlled generation.

Modality Stability shows a more modest improvement. This reflects the fact that while the pipeline strongly constrains predicate naming and structure, some ambiguity remains in mapping natural-language cues to deontic force, particularly in edge cases involving implicit obligations or strong prohibitions. Nevertheless, the pipeline avoids many of the most severe modality flips observed in the zero-shot baseline, where the same action may alternate between permission and prohibition across runs.

Overall, these metrics demonstrate that the proposed approach substantially improves the internal reliability of generated legal rules, a prerequisite for downstream reasoning, verification, and explanation. Unlike traditional accuracy metrics, Rule Consistency and Modality Stability directly measure properties that matter for formal systems: reproducibility, semantic coherence, and trustworthiness.

Chapter 6

Discussion

The results and methods described in previous sections demonstrate that enforcing structural and symbolic constraints on LLM-based formalization pipelines significantly improves the reliability of generated legal rules. While LLMs can produce potentially correct logical representations, a critical gap exists between semantic plausibility and usability. The improvements observed in Rule Consistency and Modality Stability indicate that the proposed pipeline generates representations stable enough to support downstream automated reasoning.

Discussion of Results Inconsistency, rather than a lack of expressiveness, is the primary barrier to practical LLM-to-logic systems. In zero-shot baselines, models often capture general intent but vary predicate names and argument structures across different runs. These variations are fatal for symbolic engines that require exact structural alignment to detect conflicts or apply exceptions. The near-zero consistency scores in table 5.2 of the baseline highlight the fragility of unconstrained formalization in legal contexts.

The proposed pipeline uses lightweight symbolic guidance, such as shared symbol tables and staged rewriting, to reduce variability. Grounding predicates in a stable vocabulary effectively prevents semantic drift while maintaining the flexibility of LLMs. This increase in consistency is achieved without requiring handcrafted grammars or rigid domain rules.

Modality Stability remains a significant challenge. While the pipeline constrains actors and actions, determining the deontic markers (obligation versus permission) is highly sensitive to linguistic nuance. Since legal texts often express these concepts implicitly, modality interpretation remains more difficult than predicate grounding and will need either more legal data or need more domain-specific rules to determine more accurately.

Reasoning Frameworks The use of Clingo and Z3 illustrates the trade-offs between expressive reasoning and explanatory power. Clingo's non-monotonic semantics naturally support rule overrides and exceptions. Z3 remains valuable for consistency checking and consequence explanation. Separating these concerns across tools addresses the current difficulty of unifying exception handling and model-based explanation in a single framework. As seen in appendix entry B, the clingo models for 90% of test cases are able to deduce the statement that holds accurately in meaning, with a full end-to-end example shown in appendix entry C.

Progressive Structuring Legal formalization is a process of progressive structuring rather than a single translation step. Each pipeline stage, from atom extraction to symbol grounding, adds a layer of constraint that reduces ambiguity. Treating LLMs as guided components within a symbolic system, rather than end-to-end formalizers, provides a promising direction for building trustworthy legal AI.

6.1 Future Work

While this thesis demonstrates that symbol-guided large language model pipelines can significantly improve the consistency and usability of formalized legal rules, several directions remain for future exploration. One natural extension is scaling the approach to larger and more diverse legal texts. Applying the pipeline across entire statutes or regulatory codes would introduce challenges related to cross-section references, long-range dependencies, and evolving definitions. Addressing these issues would require more sophisticated backend infrastructure, such as a graph-based representation of legal documents that explicitly models relationships between sections, definitions, and rule dependencies. Persisting symbol tables, rule graphs, and embeddings in a database-backed system, such as PostgreSQL with vector storage for semantic embeddings, would enable efficient similarity search, versioning, and incremental updates as new text is processed. Such an architecture would support scalable symbol resolution, cross-document consistency, and efficient querying over large legal text, providing a foundation for deploying the proposed pipeline beyond small, isolated examples.

Another promising direction is enriching the symbol table with stronger semantic structure. In the current system, symbols are grounded primarily through embedding-based similarity and canonicalization. Future work could integrate explicit ontological relationships between actors, actions, and legal concepts, enabling hierarchical reasoning and improved generalization across related statutes. Incorporating domain-specific legal terms or learned type systems could further enhance both precision and interpretability.

The role of human oversight also presents an important avenue for future research. Although the pipeline reduces manual encoding effort, interactive tools that allow legal experts to inspect, correct, and refine symbol mappings or rule priorities could improve trust and adoption. Such interfaces could support workflows where automated formalization is combined with targeted human validation.

From a reasoning perspective, deeper integration between defeasible logic and SMT solving remains an open challenge. While this work leverages Z3 for consistency checking and Clingo for defeasible reasoning, future systems could explore unified reasoning frameworks or hybrid solvers that natively support both explainability and rule overriding. Advances in this direction could further strengthen the reliability and transparency of automated legal analysis.

Finally, the methods developed in this thesis are not limited to legal text. Many policy-driven domains, such as organizational governance, security policies, and regulatory compliance in technical systems, share similar characteristics of exception-driven rules and normative constraints. Extending the pipeline to these domains could broaden its applicability and provide further insight into how symbolic guidance can improve the reliability of language model–based formalization systems.

Chapter 7

Conclusion

This thesis presented a structured, consistency-driven pipeline for translating natural-language legal text into formal logical rules suitable for automated reasoning. By combining large language models with explicit symbolic constraints, the proposed approach addresses a central limitation of prior work: the tendency of purely neural or purely rule-based systems to produce unstable, inconsistent, or semantically fragmented formalizations. The pipeline decomposes the formalization task into modular stages, each designed to preserve legal meaning while enforcing structural coherence.

A key contribution of this work is the introduction of a symbol-table–guided mechanism that constrains LLM outputs to a stable vocabulary of actors, actions, and predicates. This grounding step significantly reduces semantic drift across rules and across runs, enabling the generation of logical representations that are reliable enough for downstream analysis. By encoding the resulting rules in Defeasible Deontic Logic, the system captures the exception-driven and priority-sensitive nature of legal reasoning, while translations into Z3 and Clingo enable complementary forms of verification and execution. Z3 provides model-based explanations and consistency checking, while Clingo supports defeasible reasoning with explicit rule overrides.

Through representative case studies, this work demonstrates that integrating symbolic guidance with LLM-based processing leads to more coherent and analyzable formalizations than direct LLM-to-logic approaches. The resulting rules preserve statutory intent, support conflict resolution, and enable meaningful reasoning queries about permissions, prohibitions, and obligations under varying conditions. More broadly, this thesis illustrates how neurosymbolic design principles can be applied to high-stakes domains such as law, where interpretability, consistency, and correctness are as important as linguistic coverage.

While this work focuses on statutory text and defeasible deontic reasoning, the underlying ideas extend to other normative and policy-driven domains that require faithful translation from natural language to formal logic. Future directions include scaling the symbol table across larger legal corpora, incorporating richer cross-reference resolution, and developing interactive tools that allow legal experts to inspect, refine, and validate generated rule sets. By grounding language models in explicit symbolic structure, this work contributes a step toward trustworthy, machine-interpretable representations of law.

Appendix A

LLM Prompt Specifications

This appendix documents the prompts used at each LLM-assisted stage of the pipeline. The prompts are included verbatim to support reproducibility and to clarify the constraints imposed on the model during rule rewriting, segmentation, formalization, and scenario evaluation.

A.1 Baseline Zero-Shot ASP Generation Prompt

```
Convert the following legal text into Answer Set Programming (ASP) rules
using Clingo syntax.

Represent obligations, permissions, and prohibitions as ASP predicates.
Extract all actors, actions, conditions, and exceptions directly from
the text.
Use your best judgment to infer the rule structure and logical form.

Requirements:
- Rewrite the legal text into a set of ASP rules.
- Invent predicate names as needed using lowercase letters.
- Convert natural-language modality words ("shall", "may", "must not",
  "is prohibited", etc.) into ASP predicates such as:
    obligation(...)
    permission(...)
    forbidden(...)
- If the text implies conditions or exceptions, encode them using rule
  bodies.
- If multiple rules are implied, split them into separate ASP rules.
- All rules must be grounded directly from your interpretation of  the
  sentence.
- Do not ask questions.
- Output only Clingo rules.
- Clingo rules must follow the schema:
```

```
    rule(ID, modality, head_predicate).
    condition(ID, antecedent_atom).
- Output only the Clingo rules according to the schema. Do not include
  any additional text.

Input legal text:
"""
{original}
"""

Output JSON:
{
  "rules": "...",
  "explanation": "..."
}
```

A.2 Rule Rewriting Prompt

```
You are a legal analyst trained in logical interpretation.

Given this legal rule:
"""
{rule_text}
"""

Rewrite it into one or more clear, logically distinct statements that
make implicit conditions explicit.

Guidelines:
- If the rule includes phrases like "without", "unless", or "except",
  split it into two statements:
    1. A default prohibition or obligation.
    2. An exception or permission that applies when the condition is met.
- Keep the language simple, declarative, and neutral.
- Preserve the original meaning and legal intent.
- Try to keep the wording across the sentences as similar as possible.

Return JSON only in the following format:
{
  "simplified_rules": [
    "...",
    "..."
  ],
```

```
  "explanation": "..."
}
```

A.3 Rule Segmentation Prompt

```
You are a legal analyst.

Step 1: Split the following statute into distinct normative statements,
each expressing a single obligation, permission, or prohibition.

For each statement, include:
- id: r1, r2, ...
- natural_language_rule: a concise restatement in plain English.
  Restate in a way that is easier to understand if needed,
  paying attention to logical flow and structure.
- reason_role: MAIN / EXCEPTION / STRONG_EXCEPTION / CTD
- priority_over: list of rule ids that this rule overrides (if any)
- modality_hint: shall / may / shall not / must not ...

Text:
"""
{text}
"""

Output JSON:
{
  "rules": [
    {
      "id": "r1",
      "natural_language_rule": "...",
      "reason_role": "MAIN",
      "priority_over": [],
      "modality_hint": "shall"
    }
  ]
}
```

A.4 Rule Formalization Prompt

```
You are a legal logician and formal logic compiler.

Your task is to convert a natural-language rule into a strict,
```

```
machine-readable intermediate representation suitable for:

(1) Defeasible Deontic Logic (DDL)
(2) Clingo-compatible Answer Set Programming (ASP)

STRICT GLOBAL REQUIREMENTS:
- Do NOT invent new names, entities, individuals, variables, or   symbols.
- Use ONLY terms explicitly present in the rule text OR in
  the symbol table.
- Treat roles/classes (e.g., "Policyholder", "Insurer", "Physician")
  as atomic symbols.
- Never create example people such as "John", "Alice", or  placeholders.
- Never add implicit variables.
- Output must contain ONLY ASCII characters.

PREDICATE AND SYMBOL FORMAT:
- Predicate names: UpperCamelCase
- Argument names: UpperCamelCase
- No spaces, no hyphens.
- Reuse symbols from the symbol table EXACTLY.

ARGUMENT RULES:
- Arg1 = primary actor
- Arg2 = primary action object
- Arg3+ = additional context parameters
- Nested actions are allowed (e.g., Require(Insurer, Provide(...))).
- No predicate may have more than 3 arguments unless forced by the rule.

ANTECEDENT RULES:
- Antecedent must use ONLY: AND, NOT, TRUE
- No Unicode symbols.
- Each antecedent atom must be a valid predicate expression:
    Predicate(A)
    Predicate(A,B)
    Predicate(A,B,C)
- No English text inside antecedent.
- If the rule is unconditional, antecedent = "TRUE".

HEAD RULES:
- Head must be EXACTLY ONE predicate expression.
- No conjunctions or multiple heads.

MODALITY RULES:
- Modality must be one of:
```

```
    FORBIDDEN
    PERMISSION
    OBLIGATION

MAPPING GUIDELINES:
1. Identify actors, objects, and predicates ONLY from the rule text.
2. Reuse symbols from the symbol table whenever possible.
3. Convert multi-word phrases to CamelCase.
4. Do NOT infer participants.
5. Do NOT generate examples.

INPUT:
Rule (text): "{rule['natural_language_rule']}"
Modality hint: {rule['modality_hint']}
Known symbols: {symbol_table_keys}

Output JSON:
{
  "id": "{rule['id']}",
  "antecedent": "TRUE or Atom1 AND Atom2 AND ...",
  "modality": "FORBIDDEN | PERMISSION | OBLIGATION",
  "head": "Predicate(Arg1, Arg2, Arg3)",
  "explanation": "Short explanation of how the structure was derived."
}
```

A.5 Scenario Condition Evaluation Prompt

```
You are a reasoning assistant.

Your task is to determine which conditions are TRUE or FALSE in a test
scenario based on:

1. The natural-language scenario.
2. The Clingo-style rule encoding provided.

Rules describe permissions and prohibitions, but your task is NOT to
decide legal validity. Your ONLY job is to infer which CONDITION
predicates should be marked true or false based on the scenario
description.

-------------------------------------
INPUT FORMAT
-------------------------------------
```

```
ORIGINAL SCENARIO:
{original}

RULES (Clingo-style):
{rules}

SCENARIO:
{scenario_text}

-------------------------------------
YOUR TASK
-------------------------------------

1. Read the scenario.
2. Look at every condition(...) appearing in the rules.
3. For each condition, output a TRUE or FALSE value based ONLY on
   what the scenario explicitly states.
   - If the scenario clearly supports the condition → TRUE
   - If the scenario contradicts it → FALSE
   - If the scenario gives no information → FALSE
4. Do NOT infer unstated facts.
5. Do NOT output explanations or restate rules.

-------------------------------------
OUTPUT FORMAT (STRICT)
-------------------------------------

<condition_name> = True/False
(one per line)
```

Appendix B

Scenario-Based Reasoning Outputs

This appendix presents the raw outputs produced by the Clingo-based defeasible reasoning backend for ten scenario-based test cases. Each example consists of (i) an original policy statement written in natural language, (ii) a concrete scenario description, (iii) the derived condition assignments used as input facts, and (iv) the resulting deontic conclusions computed by the solver.

For each example, the pipeline first formalizes the policy text into Defeasible Deontic Logic rules, then assigns truth values to condition predicates based solely on the scenario description, and finally executes the resulting logic program in Clingo. The `result` field reports the final normative conclusions (permissions, prohibitions, or obligations) that hold after accounting for rule applicability and priority-based overrides.

Example 1

Policy. Restricted park zones are off-limits to civilians. Park rangers have routine access. Firefighters are allowed entry only when a wildfire alert is active. Regardless of role, anyone with an open safety violation cannot enter.
Scenario. A firefighter with a clean record attempts to enter during an active wildfire alert.
Result.
`permission(enter_restrictedparkzones)`

Example 2

Policy. Only staff members may freely use staff computers. Volunteers may use them as long as a staff member is supervising. Guests are not permitted. Anyone banned from the library loses all computer privileges.
Scenario. A volunteer sits beside a supervising librarian and logs into a staff computer.
Result.
`permission(allow_staffcomputers)`

Example 3

Policy. Dispensing controlled medication is limited to licensed medical personnel: registered nurses and doctors. Doctors may do so at their discretion, while nurses may act only if not under disciplinary review. Patients cannot dispense medication under any circumstances.
Scenario. A registered nurse with no active review prepares controlled medication for a patient.
Result.

```
obligation(dispense_controlledmedication)
```

Example 4

Policy. Active construction zones may be accessed only by certified workers. Inspectors are allowed entry if a site manager accompanies them. Wearing a hard hat is mandatory for anyone entering construction areas.
Scenario. A certified worker wearing all required gear walks into the active zone.
Result.

```
forbidden(enter_activeconstructionzones)
obligation(wear_hardhat)
```

Example 5

Policy. High-voltage equipment can be operated by graduate researchers and faculty. Students are not permitted. Completion of the electrical safety module is required for all operators.
Scenario. A graduate researcher who finished the safety module turns on the high-voltage unit.
Result.

```
permission(operate_highvoltageequipment)
obligation(complete_safetymodule)
```

Example 6

Policy. Runway access is restricted to ground crew. Security officers may enter only when the airport is in a heightened alert status. A valid airport-issued badge is required for anyone entering the runway.
Scenario. A ground crew technician with a valid badge drives onto the runway.
Result.

```
permission(access_runway)
```

Example 7

Policy. Confidential datasets may be accessed by researchers and project leads. Interns are never permitted. Anyone who has previously violated data policy loses access privileges.
Scenario. An intern requests the confidential dataset.
Result.
`forbidden(access_confidentialdatasets)`

Example 8

Policy. Municipal maintenance vehicles may be operated only by certified technicians. Emergency operators may operate them during declared severe weather. Operators with suspended licenses are prohibited.
Scenario. A licensed, certified technician starts a municipal utility vehicle on a normal day.
Result.
`(no normative conclusions)`

Example 9

Policy. Editing school network settings is allowed for IT staff and administrators. Students may not modify network configurations. Any attempt using an unregistered device is automatically prohibited.
Scenario. An IT technician working on a registered device updates the network router.
Result.
`permission(edit_networksettings)`

Example 10

Policy. Only shelter staff may approve adoption paperwork. Veterinarians may authorize paperwork only for medical-case animals. Walk-in visitors cannot approve anything. Staff members under misconduct investigation are barred from approvals.
Scenario. A staff member with no investigations pending signs off on an adoption.
Result.
`obligation(approve_adoptionpaperwork)`

Appendix C

End To End Example

Policy Text.

> Restricted park zones are off-limits to civilians. Park rangers have routine access. Firefighters are allowed entry only when a wildfire alert is active. Regardless of role, anyone with an open safety violation cannot enter.

Encoded Rules. The policy text is formalized into the following defeasible rules. Rule identifiers are used to express priority relationships for exception handling.

```
rule(r1, forbidden, enter_restrictedparkzones).
condition(r1, civilians).

rule(r2, permission, enter_restrictedparkzones).
condition(r2, parkrangers).

rule(r3, permission, enter_restrictedparkzones).
condition(r3, firefighters).
condition(r3, wildfirealertactive).

rule(r4, forbidden, enter_restrictedparkzones).
condition(r4, opensafetyviolation_anyone).

overrides(r4, r2).
overrides(r4, r3).
```

Scenario Description. A firefighter with a clean record attempts to enter during an active wildfire alert.

Scenario Facts. From the scenario description, the following condition predicates are inferred. Conditions not explicitly supported by the scenario default to false.

```
holds(firefighters).
holds(wildfirealertactive).
```

```
not_holds(civilians).
not_holds(parkrangers).
not_holds(opensafetyviolation_anyone).
```

Solver Output. Executing the Clingo program with the above rules and facts yields the following normative conclusion:

```
permission(enter_restrictedparkzones)
```

Interpretation. In this scenario, the firefighter-specific exception applies because the wildfire alert condition holds. No higher-priority prohibition is triggered, as the individual does not have an open safety violation. As a result, the general civilian restriction is overridden, and entry into the restricted park zone is permitted.

Appendix D

Code Availability

To support reproducibility and facilitate further research, the full implementation of the pipeline described in this thesis is publicly available online. The repository includes source code for legal text preprocessing, symbol-table construction, rule segmentation, logical formalization, and Clingo-based reasoning, along with scripts used to generate the experimental results reported in this work.

The codebase is hosted on GitHub at:

`https://github.com/spanthan/masterthesiscode.git`

The repository also contains instructions for setup and execution, example inputs, and configuration files used in the experiments.

Bibliography

[1] Sam Bayless, Stefano Buliani, Darion Cassel, Byron Cook, Duncan Clough, Rémi Delmas, Nafi Diallo, Ferhat Erata, Nick Feng, Dimitra Giannakopoulou, Aman Goel, Aditya Gokhale, Joe Hendrix, Marc Hudak, Dejan Jovanović, Andrew M. Kent, Benjamin Kiesl-Reiter, Jeffrey J. Kuna, Nadia Labai, Joseph Lilien, Divya Raghunathan, Zvonimir Rakamarić, Niloofar Razavi, Michael Tautschnig, Ali Torkamani, Nathaniel Weir, Michael W. Whalen, and Jianan Yao. A neurosymbolic approach to natural language formalization and verification. Technical report, arXiv, 2025. arXiv:2511.09008. 2.3

[2] Mauro Dragoni, Leon Van der Torre, and Serena Villata. Combining nlp approaches for rule extraction from legal texts. In *Proceedings of the 31st International Conference on Legal Knowledge and Information Systems (JURIX 2018)*. IOS Press, 2018. 2.1

[3] Evan Horner, Christian Mateis, Guido Governatori, and Agata Ciabattoni. From legal texts to defeasible deontic logic via llms: A study in automated semantic analysis. Technical report, TU Wien and Austrian Institute of Technology and Charles Sturt University, 2025. 2.2