Mnemosyne: Using Large Language Models to Convert Documents to Knowledge Graphs to Check for Completeness and Consistency

by Michael Wacey

B.S. in Computer Science, June 1985, Washington University in St. Louis
M.S. in Computer Science, December 1990, Drexel University

A Praxis submitted to

The Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial satisfaction of the requirements
for the degree of Doctor of Engineering

May 17, 2026

Praxis directed by

Mohamed Elbasheer
Professorial Lecturer in Engineering and Applied Science

The School of Engineering and Applied Science of The George Washington University certifies that Michael Wacey has passed the Final Examination for the degree of Doctor of Engineering as of November 24, 2025. This is the final and approved form of the Praxis.

Mnemosyne: Using Large Language Models to Convert Documents to Knowledge Graphs to Check for Completeness and Consistency

Michael Wacey

Praxis Research Committee:

John M. Fossaceca, Adjunct Professor of Engineering and Applied Science, Chair

Mohamed Elbasheer, Professorial Lecturer in Engineering and Applied Science, Advisor

Jonathan Ng, Professorial Lecturer of Engineering and Applied Science, Committee Member

Dedication

In loving memory of my parents, Jack and Helen. I dedicate this work to their belief in me and the lessons they taught. From my mother, I learned what is possible through sheer determination. The memory of her returning to school to earn a master's in economics while raising our family is an inspiration to me. From both of my parents, I received the encouragement and love of learning that made this achievement possible.

To my siblings, Gordon, Carole, and Iain, who have graciously allowed me to neglect them during this endeavor. I am deeply grateful for your patience and look forward to getting back together now that it is complete.

To my children, Alex, Peter, and Kayla. Your support and understanding throughout this process have been a unending source of strength for me. Thank you for providing me the support I used to see this through.

And most importantly, to my lovely wife, Elaine. The conversation was not a long one. As soon as I found this program you encouraged me to start. During the first year when I had to get up early or sta up late you made sure that I had coffe and anythig else I needed. I am grateful for all that you have done.

Acknowledgments

I would like to express my profound gratitude to my advisor, Dr. Elabasheer, whose mentorship was instrumental in guiding this research. He showed me the way forward when the path was unclear and challenged me to think more critically. Whenever I find myself thinking I cannot multitask or overcome a challenge, I will undoubtedly hear his voice reminding me that I must—a lesson in perseverance I will carry throughout my career.

My deepest appreciation extends to all my professors at The George Washington University who taught me during this program. Their dedication to teaching and intellectual generosity created the rich academic environment and provided the abundant knowledge that were essential for this journey.

I am also deeply grateful to my West Chester University Dean, Dr. Burns, who served as a steadfast source of support. He was always available to listen to my trials and tribulations, and with a few short, insightful words, he consistently helped me untangle complex problems and understand how to proceed. His wisdom and perspective were invaluable.

A special and heartfelt thank you is reserved for my wife, Elaine. As a lawyer, her tolerance for dense text is matched only by her sharp eye for detail—both of which were indispensable gifts to this project. She patiently read every single word at least ten times, and her contribution to this work is immeasurable.

Finally, I wish to acknowledge my students. Their intellectual curiosity and enthusiasm for learning were a constant source of motivation. It was my desire to be a more effective and knowledgeable teacher for them that provided the final impetus to see this project through to its completion. You reminded me daily of why this work matters.

Abstract

Mnemosyne: Using Large Language Models to Convert Documents to Knowledge Graphs to Check for Completeness and Consistency

It is difficult for people to maintain large complex documents like township ordinances or technical manuals. The review process is slow and costly and is prone to human error. This lets internal inconsistencies and incompleteness make their way into documents. That can trigger legal disputes and ultimately financial losses. Large Language Models (LLMs) would seem to be an obvious solution to this. However, all LLMs have a finite context window and therefore cannot process an entire large document.

This Praxis details the design, creation, use, and evaluation of Mnemosyne. It is a system built to overcome the LLM context window limitation. Mnemosyne makes use of LLMs to convert large documents into attributed knowledge graphs (KGs). It does this by breaking the document into small chunks. It has a multi stage pipeline that uses a two tiered memory architecture. Through this design the system can incrementally build and iteratively refine the knowledge graph. In doing so, it creates a KG that faithfully represents the full semantics of the document. Mnemosyne's was tested using the ordinances of Pennsylvania townships. These test included controlled injection of related material and unrelated material. Another test involved the removal of a section.

The results demonstrate that Mnemosyne can construct useful knowledge graphs from township ordinances. These knowledge graphs retain the ability to trace back to the source documents. Experiments showed that the system can be used to identify inconsistencies. These appear as irregularities in the knowledge graph's structure.

A key contribution of this research is a two part definition of document completeness. The experiments show that a knowledge graph is effective at detecting internal completeness issues. For example, undefined terms and broken cross references. This is done through simple structural analysis. However, the research also establishes that external completeness cannot be detected by internal analysis alone, it requires external domain knowledge. This Praxis validates that an LLM approach to building and refining an attributed knowledge graph provides a practical and robust methodology for verifying the internal consistency and structural completeness of large documents.

Table of Contents

List of Figures

List of Tables

Glossary of Terms

AI Artificial Intelligence

API Application Programming Interface

BERT Bidirectional Encoder Representations from Transformers

BPE Byte Pair Encoding

C&C Consistency and Completeness

CRF Conditional Random Field

CTSA Conewago Township Sewer Authority

EDA Exploratory Data Analysis

EDR Error Detection Rate

ET Easttown Township code

GNN Graph Neural Network

GPT Generative Pre-trained Transformer

GPU Graphical Processing Unit

IE Information Extraction

JSON JavaScript Object Notation

KG Knowledge Graph

LIR Legal Information Retrieval

LLM Large Language Model

LTM Long-Term Memory

NER Named Entity Recognition

NLP Natural Language Processing

OpenIE Open Information Extraction

OWL Web Ontology Language

RAG Retrieval-Augmented Generation

RDF Resource Description Framework

RE Relationship Extraction

SHACL Shapes Constraint Language

SPARQL SPARQL Protocol and RDF Query Language

STM Short-Term Memory

W3C World Wide Web Consortium

Chapter 1: Introduction

1.1 Background and Research Motivation

It has been identified that verifying the completeness, consistency, and correctness of large and complex documents is an important task (Zowghi & Gervasi, 2003). Correctness often depends on external knowledge, while completeness and consistency can be assessed by looking at the document itself. This research focuses on these aspects. It proposes an automated tool that makes use of large language models (LLMs) to transform lengthy documents into knowledge graphs (KGs). By distilling the content of a document or set of documents into an integrated set of KGs, reasoning about the completeness, consistency, and correctness of the underlying documents can be accomplished.

As artifacts such as legal documents, technical manuals, and regulatory frameworks grow in size and scope, manual review becomes increasingly challenging. Manual processes are labor-intensive and tend to be error prone. Recent advances in artificial intelligence (AI) have provided powerful tools that can be used for text analysis. In particular, transformer-based LLMs. A well-understood limitation of these models is that they have a fixed context window that can be small for certain tasks. This prevents them from fully processing a long document. To address this limitation, the present effort utilizes KGs to model the full semantic content of a document or set of documents. KGs are a structured representation of a source. The source is decomposed into entities, their relationships, and attributes. The LLM can then be used to reason about topics in the KG that are semantically related. This approach is applied to Pennsylvania township laws, where inconsisten-

cies within the laws can cause legal disputes and significant financial losses.

The motivation for this work arose from the inconsistencies that crept into the ordinances of Easttown Township over many years. In Pennsylvania, new local regulations are compiled into existing ordinances by hand. This manual process frequently introduces inconsistencies or leaves the ordinances incomplete. There is a need for a solution, preferably automated, that addresses this problem. The cost of doing it manually is too much to bear for most townships. The goal of this research is to leverage the language understanding capabilities of LLMs to address the need to ensure the integrity of large documents. In the Eattown case, the Subdivision Laws required payment of a fee in lieu for sidewalks not built. Meanwhile, the Zoning laws have been changed over time to remove any capability to require payment.

1.2 Problem Statement

Municipal laws in Pennsylvania townships, authored by multiple individuals over time, develop inconsistencies and incompleteness (Curley, 2024; Rau, 2024; Sanders, 2024), which have led to annual revenue losses of hundreds of thousands of dollars (Bosco, 2024).

The complexity of these municipal ordinances is a direct result of their incremental and decentralized development. Over many years, as different officials, legal counsel, and staff draft and amend ordinances, these ordinances can become fragmented. It is common for no one to notice the unintended consequences of a new ordinance or a change to an existing ordinance. This results in overlapping provisions throughout the documents. The language becomes contradictory within and between documents. Regulatory gaps also emerge. Townships will face issues if they do not have a

way to address these problems. For example, townships may not be able to enforce their laws, and they risk losing revenue.

Inconsistencies in ordinances will have financial consequences for the governments involved. Ambiguous zoning regulations could allow for land development without the collection of fees. Unclear tax ordinances can lead to disputes between the government and residents. This will result in reduced tax collections. These problems can strain local budgets. Then, public services will have to be cut. To address this issue, an automated and streamlined approach to analyzing new and changed ordinances is required. The current labor-intensive process is inadequate for this task.

1.3 Thesis Statement

This praxis demonstrates that Mnemosyne, an LLM-based tool designed to convert documents into attributed knowledge graphs, can be used to check for consistency and completeness, which will allow municipal lawyers to create complete and consistent legal documents, thereby preventing costly disputes and reducing revenue losses.

Mnemosyne is named after the Greek Titan who was also the goddess of memory. The name reflects the function of transforming unstructured text into structured memory. The structured memory is the KG. It also provides an indication of the system's internal architecture (detailed in Chapter 3). This architecture uses ”short-term” and ”long-term” memory components to incrementally build the KG.

The conversion of documents into KGs makes the analysis of large documents both easier and more tractable. A human reviewer may struggle to trace dependencies across hundreds of pages. A KG creates a map of all the entities and their relationships from a document. Mnemosyne builds upon

this by creating a rich ”type” and ”part of” hierarchy on top of the KG. It is this rich hierarchy that allows the analysis to be focused, rather than having to pay attention to the full document. For example, if one is looking for issues related to a new law on noise, the focus can be on types related to noise. In doing so, the number of entities and relationships that need to be analyzed is greatly reduced.

A KG allows for the analysis of specific concepts within a document. That is focused on a specific set of related entities in the KG. For legal documents, this supports the finding of contradictions or gaps. This leads to proactive validation that:

- minimizes legal ambiguity
- improves the clarity and coherence of municipal codes
- increases the efficiency of the entire legal review process.

Inconsistencies in ordinances can have significant negative financial impacts. Therefore, it is important to identify and resolve these inconsistencies as early as possible. Preferably during the drafting process. This allows municipalities to prevent costly litigation and avoid revenue loss. The use of LLM-based tools in this way represents a step toward more modern local governments. It offers a scalable solution to maintain ordinances.

1.4 Research Objectives

The goal of this praxis is to build a tool that can automatically process a document of any size. In the first step, the document is processed and used to create a KG. To do this, the tool will extract named entities, relationships, and attributes. These will be considered a complete representation

of the source text. The resulting KG will be analyzed to determine whether it is suitable for checking the internal consistency and completeness of the source document. This process will be demonstrated by intentionally introducing targeted errors into source documents. Experiments will then be conducted to identify the introduced errors through queries on the graph representation.

1.5 Research Questions

This work addresses the following questions in order to meet the research objectives:

RQ1: Can an LLM be used to convert a large document into a knowledge graph?

RQ2: Can an LLM be used to process multiple knowledge graphs into a typed cluster of knowledge graphs?

RQ3: Can a typed cluster of knowledge graphs be used to check a source document for consistency and completeness?

1.6 Research Hypotheses

This research tests the following hypotheses:

H1: An LLM can be used to convert a large document into a knowledge graph.

H2: An LLM can be used to process multiple knowledge graphs into a cluster of typed knowledge graphs.

H3: A typed cluster of knowledge graphs can be used to verify the consistency and completeness of the source document.

1.7 Research Scope and Limitations

1.7.1 Research Scope

Pennsylvania township laws serve as the primary document source for developing and testing the proposed methodology. These publicly available documents were chosen because their complexity and collaborative authorship make them a perfect basis. While this research primarily focuses on legal documents, the methodology is designed to be usable in any domains that use large, structured documents. The central aim of this research is to build KGs that accurately represent the document's content. Here, their suitability for consistency and completeness checking is validated. A full implementation of automated validation is left for future research.

1.7.2 Research Limitations

This study has several limitations. System performance heavily depends on the underlying LLM's performance. The document chunking strategy also carries the risk of losing context. This research uses existing LLMs rather than developing a new specialized model. The evaluation relies on public documents, which are treated as a gold standard. Finally, the work is limited to processing text in Microsoft Word format. This is due to the use of paragraphs in the chunking strategy. Some formats, such as PDFs, present complexities in finding paragraph boundaries.

1.8 Praxis Organization

The remaining chapters of this praxis are organized as follows. In Chapter 2, a summary of the most relevant literature used is presented. While

over 200 references were covered, fewer are included here. The methodology to develop, run, and test Mnemosyne is presented in Chapter 3. The results of running Mnemosyne and analyzing the information can be found in Chapter 4. Chapter 5 discusses the findings from the previous chapters, their contributions, and recommendations for future research.

Chapter 2: Literature Review

2.1 Introduction

The artificial intelligence (AI) landscape was driven forward by the work conducted at Google Brain. This was presented in the influential paper Attention Is All You Need (Vaswani et al., 2017). This paper introduces a new approach called the transformer architecture. The architecture leverages self-attention mechanisms and serves as the foundation for modern large language models (LLMs). These LLMs have demonstrated remarkable capabilities across many tasks. This includes text generation, summarization, translation, and question answering. An LLM will often produce output that is nearly indistinguishable from human writing (Badshah & Sajjad, 2024; Verma, 2024).

Despite these advances, LLMs still have, and will always have, an architectural limitation regarding their context window. This window represents the maximum number of input tokens that a model can process when generating a response. Therefore, if there is information in a document on either side of this context window, the LLM will not be able to identify any relationships. This happens when the information is separated by a length of text that is greater than the LLMs context window (Kaplan et al., 2020). The limitation presents a challenge when analyzing large documents. Since understanding may depend on the ability to address information on both sides of the context window.

A good way to address this limitation is to transform documents into knowledge graphs (KGs). The first step is to extract entities, their relationships, and associated attributes from the text. Then, map them into a graph

structure. With this structure, it is possible to represent a document's content in a format suitable for analysis (Hogan et al., 2021). A KG allows for reasoning across the entire document by focusing on closely related entities. In this way, it addresses the problems caused by an LLM's context window limitations. Thereby enabling an LLM to focus on semantically related portions of the document. This will support completeness and consistency checks.

This chapter reviews the relevant literature for this approach. First, an examination of the development and resulting characteristics of LLMs is conducted. The focus is on their capabilities and limitations. Particularly, the context window constraint. After this, there is an exploration of the principles, construction, and application of KGs structured knowledge representation. Key techniques for turning text into KGs are explored. These include information extraction (IE) and Relationship Extraction (RE). the next discussion covers challenges associated with processing large and complex documents. A focus on legal document issues is included. The chapter conclusion begins by defining the core concepts of this praxis: consistency, completeness, and coherence. Related work is surveyed. Lastly, there is a summary of the motivations that led to this research.

2.2 Large Language Models

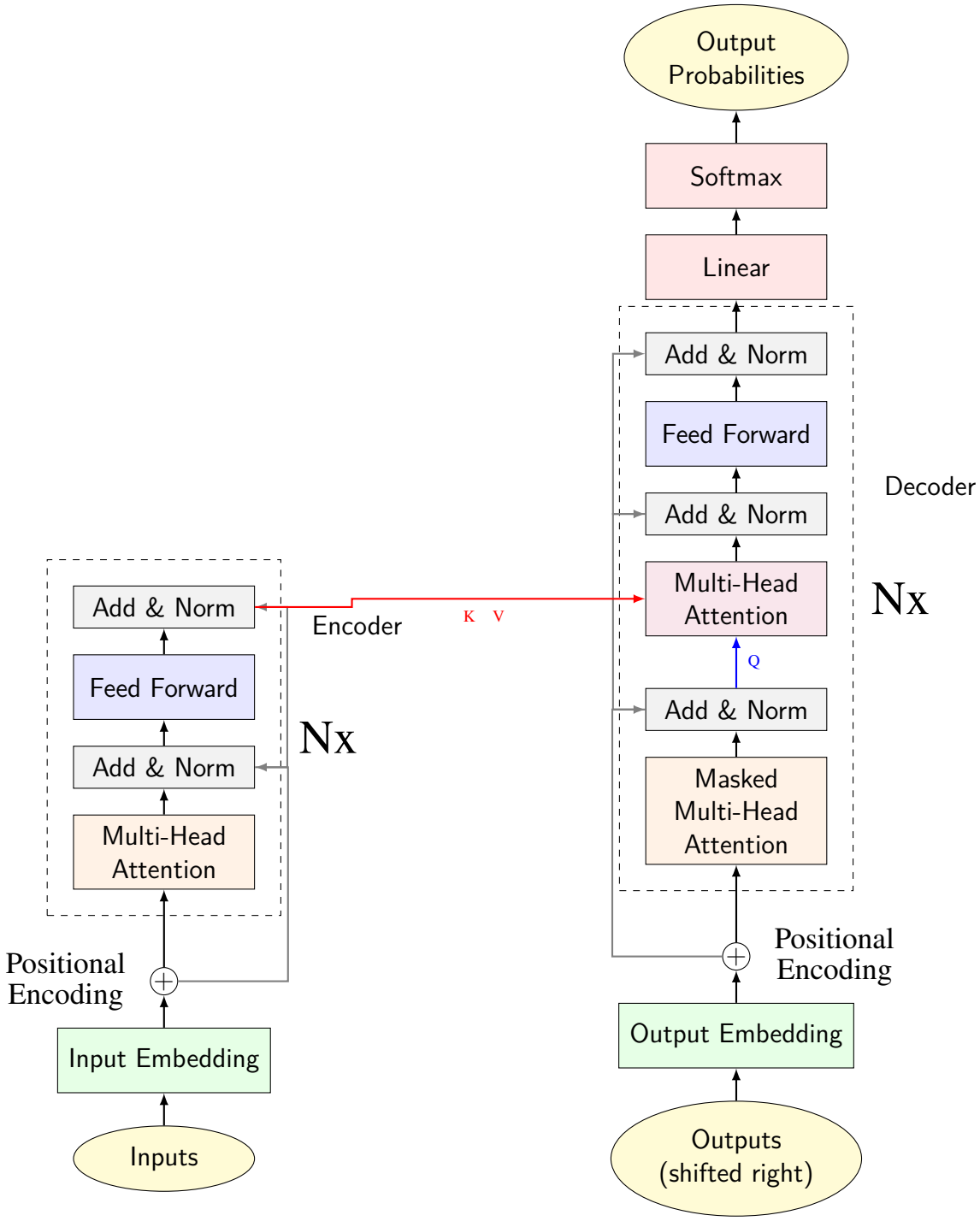
The trajectory of modern Natural Language Processing (NLP) shifted in 2017 with the publication of Attention Is All You Need by (Vaswani et al., 2017). This work introduced the transformer architecture. This architecture relies on self-attention mechanisms to weigh the importance of different words (tokens) in an input sequence. This design enables handling of long-range dependencies compared with earlier architectures of recurrent or

convolutional networks. Thereby addressing the critical bottlenecks in sequence modeling (Turner, 2024; Zhao et al., 2023). This innovation set the stage for the development of increasingly powerful large language models. For example, Google's BERT, which introduced bidirectional pre-training (Gardazi et al., 2025). Also OpenAI's influential Generative Pre-Trained Transformer (GPT) series (Gao et al., 2023).

Research groups across numerous institutions continued to pursue improvements. OpenAI elected to publicly release ChatGPT on November 30, 2022. This dramatically increased public awareness of advanced conversational AI systems. It also catalyzed the development of competing models from major research laboratories, including Google's Gemini family, Anthropic's safety-focused Claude series, and Meta's open-source Llama family (Caruccio et al., 2024; Grattafiori et al., 2024; Team et al., 2024). The proliferation of models is evident on platforms such as Hugging Face, a central repository for AI models and datasets, which reportedly surpassed one million hosted models by late 2024 (Edwards, 2025).

Functionally, LLMs process input text (a "prompt") by converting it into numerical representations called tokens. They do this with techniques such as byte-pair encoding (BPE) or WordPiece (Schmidt et al., 2024). The model then employs the complex patterns learned during pre-training on vast sets of text to autoregressively predict subsequent tokens. This results in the generation of coherent and contextually relevant output. Prompts can be engineered to elicit specific behaviors. This includes the analysis of substantial text provided for context (in-context learning). For instance, an LLM can be prompted with a company's annual report to answer specific questions or summarize key findings. These are tasks that many current models perform reasonably well. That is, provided the information falls within their

processing limits (Rzepka et al., 2023). The architecture of the transformer model is depicted in Figure 2.1.



adapted from Vaswani et al. (2017)

Figure 2.1: *The Transformer - model architecture*

A fundamental limitation of LLMs is the context window size. This represents the maximum number of tokens to which the model can attend simultaneously. It is always finite. Although it has increased with newer model generations (Kaplan et al., 2020; Liu et al., 2025; Ratner et al., 2022). If a document's length exceeds this limit, the LLM cannot process it in a single pass. Standard techniques to address this limitation involve processing the document in chunks. This approach can sever long-distance contextual links crucial for deep understanding (T. Chen et al., 2024). For example, determining if a policy defined on the first page of township ordinances is contradicted by regulations hundreds of pages later can be impossible. This is because, if the text between them exceeds the context window, the model would have to process these sections independently.

The growth in the computational cost of processing text within the context window is a difficult problem to address. The self-attention mechanism of transformers scales quadratically $(O(n^2))$ with the input sequence length (n). This includes both computation and memory requirements (Vaswani et al., 2017). There are some "efficient transformer" variants that try to reduce this complexity. A few even approach linear growth. Even so, the processing of long sequences still demands substantial resources (Tay et al., 2023). This growth makes the analysis of large documents prohibitively expensive for practical applications. This leads to the search for alternative approaches. For example, KG-based structuring. The expectation is that this will achieve comprehensive and efficient analysis.

2.2.1 The Challenge of Long-Context Processing

The transformer architecture revolutionized NLP. It is based on the self-attention mechanism. This mechanism has a significant scalability problem.

It has a quadratic growth in computational and memory costs based on the length of the input sequence (Vaswani et al., 2017). This creates an upper limit on the context window's size, and at some point, it becomes too large to reasonably process the contents when full. This poses a major hurdle for using the transformer architecture to process lengthy documents. Researchers have applied several strategies to mitigate this limitation. For example, architectural innovations such as sparse-attention mechanisms (e.g., Longformer, BigBird) modify the attention matrix to attend to only a subset of tokens. This reduces the computational complexity from quadratic to near-linear (Beltagy et al., n.d.; Zaheer et al., n.d.). Other approaches include Transformer-XL. It introduces a recurrence mechanism that is used to cache hidden states from previous segments. This allows information to flow beyond a fixed-size window (Dai et al., 2019).

A more recent and widely adopted approach is the use of retrieval-augmented generation (RAG) (Lewis et al., 2020). In a RAG system, a large set of documents is first segmented into chunks and then indexed in a vector database. When a query is posed. The system retrieves the most semantically relevant chunks and injects them into the LLM's context window. This serves as supplementary information for generating a response. This is highly effective for focused, knowledge-intensive tasks. For example, answering a specific question. RAG pulls specific document fragments that are related to the current prompt. It inherently limits the amount of information retrieved so as not to overwhelm the context window. It is less suited for tasks requiring a holistic, structural understanding of a document. Such as verifying global consistency. Suppose the need is to detect whether a term defined in Chapter 1 is used in a contradictory manner in Chapter 20. A RAG system would have to coincidentally retrieve both the specific and the distant

chunks in the same query. It is unlikely to do this. This praxis contends that for comprehensive document validation, a pre-constructed KG, which explicitly models the entire document's entities and relationships, offers a more robust and queryable structure than the on-the-fly, fragmented context provided by RAG.

2.3 Knowledge Graphs

KGs evolved from concepts in semantic networks, frame systems, and earlier AI research in symbolic knowledge representation (Hogan et al., 2021). KGs provide a structured paradigm for representing information. Formally, a KG represents knowledge as a directed, labeled graph. They are constructed of a collection of interconnected entities (nodes) and explicitly typed relationships (edges) between them. Both nodes and edges can possess attributes or properties that store additional metadata or context (Cong, 2024; Ehrlinger & Wöß, 2016).

The core components of a KG are as follows:

- **Nodes (entities):** These represent real-world objects, abstract concepts, events, or specific instances of interest. Examples include persons, organizations, locations, legal statutes ("15 Pa.C.S.A. § 1502"), or defined terms ("nonconforming use"). Nodes are often identified by unique identifiers.

- **Edges (relationships):** These represent the typed relationships between pairs of nodes, such as "works for," "located in," "cites," or "amends." Edges are typically directed from a subject node to an object node and are labeled with the type of relationship.

- **Attributes (properties):** These are key-value pairs that can be on the

nodes or the edges. They provide additional details. For instance, a "Person" node might have an "email" attribute, or a "cites" edge might have an "effectiveDate" attribute.

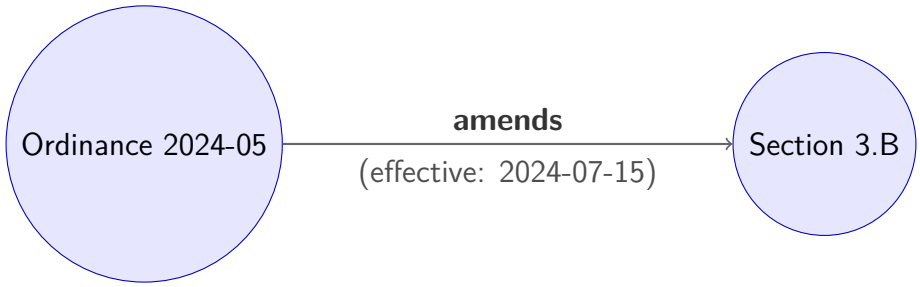


Figure 2.2: *Knowledge graph fragment of a legal amendment*

Early work on structured knowledge is Minsky's concept of frames. Frames represent knowledge using frames that contain slots. A slot can contain a value or a relationship to another frame. For example, a car frame could have a value slot for price that contains the price and a relationship slot for color that points to a blue frame. This work has influenced subsequent formalisms, such as description logics and semantic web ontologies (Minsky, 1974).

2.3.1 Ontologies and Schema in KG Construction

A KG's can be created dynamically solely from the information that is used to build it. Alternatively, it can be built based on a schema in addition to the information. A schema in this case can also be referred to as an ontology. An ontology is a hierarchy of entities that can drive the resulting structure of the KG. An ontology serves as the conceptual framework for the knowledge domain, explicitly defining the types of entities (classes), their attributes (properties), and their permissible relationships (Noy & McGuinness, 2001). For legal documents, an ontology might define classes such as Ordinance,

DefinedTerm, and ZoningDistrict. It also includes rules that an Ordinance can have an enactmentDate property of type Date. Or that an Ordinance can be linked to a DocumentSection via an AMENDS relationship.

This formal specification in the ontology is not just the start of building the KG. It creates a path for the automated validation of the KG. The establishment of a well-defined schema allows both consistency and completeness to be programmatically verified. Consistency is checked by identifying graph structures that violate ontological rules. For example, if a node representing a township is linked to a PermittedUse node, this would violate a schema rule stating that only ZoningDistrict nodes can have such a relationship.

Completeness can be assessed using constraints defined in the schema. For example, the ontology may specify that every DefinedTerm node must have at least one incoming DEFINES relationship. It may also specify that the relationship must originate from a DocumentSection node. So, any DefinedTerm node that is missing this relationship is considered incomplete. Finding violations in a KG indicates that there is a flaw in the source document. With these rules, the ontology transforms the KG from a simple data structure into a formal model that can be used to determine the integrity of the document.

2.3.2 Reasoning Over Knowledge Graphs

KGs are more than just a representation of the documents from which they are built. They are structures that can be used to perform an analysis of the document content. For example, it is possible to discover relationships in a KG that are not explicitly found in the source text. The use of a KG to represent the text in a document allows for comprehensive analysis. This

can be done through either deductive or inductive reasoning.

Deductive reasoning is a process in which new facts are derived from established facts. This is done with a set of logical rules and axioms. The rules and axioms are defined in the ontology. The rules and axioms can be used to derive new conclusions.

There are several languages that do this. The Web Ontology Language (OWL) is one technology. Another is the Shapes Constraint Language (SHACL) (Hitzler et al., 2009; Knublauch & Kontokostas, 2017). For example, if the KG contains the facts (Ordinance_123, AMENDS, Section_3.B) and (Section_3.B, PART_OF, Article_III). A deductive reasoner could apply a transitivity rule to infer the new triple (Ordinance_123, IMPACTS, Article_III). This capability is crucial for the system to fully understand the document. In doing so, it is more likely that issues will be identified.

Inductive reasoning is stochastic and can be done with machine learning models. These models are designed to operate on structured graphs. An example of such structures is graph neural networks (GNNs) (Gupta et al., 2021). A primary application of inductive reasoning in KGs is link prediction. The machine learning model is taught patterns from an existing graph structure. It is then able to predict the likelihood of missing edges between nodes. In document analysis, link prediction could identify a likely but absent USES_TERM relationship. This could be between a text section and a corresponding defined term. This could be used to flag a potential omission for human review. While not logically conclusive, this inductive approach provides a powerful tool for identifying gaps.

2.3.3 KG Technologies and Applications

KGs are implemented using a variety of technologies. The Resource Description Framework (RDF) is based on triples (subject–predicate–object) and is a W3C standard. RDF is the foundation of the Semantic Web. The language SPARQL is often used to query RDF. The KGs can be defined with ontology languages such as OWL (Hitzler et al., 2009; N. Kumar & Kumar, 2013; "RDF 1.2 Primer," 2025). The graph database, Neo4j, supports the storage of KGs in a structure known as Property KGs. This offers a flexible model to store nodes, edges, and associated properties. They can be queried with languages such as Cypher or Gremlin (Fernandes & Bernardino, 2018). In addition, the machine learning technique of graph neural networks (GNNs) operates on graph structures. They learn vector representations (embeddings). This enables tasks such as link prediction and node classification (Gupta et al., 2021; Scarselli et al., 2009; K. Wang et al., 2024).

KGs have been employed in many applications. These include search and recommendation systems, as well as enterprise data integration (Fensel et al., 2020; Hogan et al., 2021; Ji et al., 2022). They have the ability to explicitly model complex relationships. This makes them valuable for analyzing the internal structure and integrity of large documents.

2.4 Information Extraction for KG Construction

When a KG is used for document analysis, it requires multiple steps. These are the steps that take the text in a document and transform it into a KG. Collectively, these steps are part of a process called KG construction. This relies on information extraction (IE) techniques (Kolluru et al., 2020;

Zhong et al., 2024). This section covers two of the IE tasks. First, the identification of nodes via named-entity recognition (NER). Second, the identification of edges via relation extraction (RE) is essential. LLMs have shown significant promise in the performance of both tasks. A really nice feature is that it requires minimal task-specific training data (Benjira et al., 2025).

2.4.1 Named-Entity Recognition

As discussed above, (NER) is a part of information extraction. It is used to find and classify named entities in the text (Al-Moslmi et al., 2020). These categories can include standard types such as a person or an organization. In addition, they can be extended to domain-specific types. NER is crucial in identifying these entities that become the nodes in the construction of a KG. A subsequent step, entity linking, is often necessary to disambiguate these entities and link them to unique identifiers (Chaurasiya et al., 2022).

Methods for NER have evolved significantly. Early approaches were rule-based systems that used handcrafted rules and dictionaries. They achieved high precision but proved brittle and labor-intensive (Grishman & Sundheim, 1996; Nadeau & Sekine, 2007). These were followed by several statistical models. An example of this is the Hidden Markov Model. Another important statistical model is conditional random fields (CRF). This offered a better generalization (Lafferty et al., 2001). Currently, the approach of deep learning, particularly transformer-based models such as BERT, has achieved almost human comparable performance by leveraging powerful pre-trained representations (Al-Moslmi et al., 2020; Carbonell et al., 2020).

Applying NER to the legal domain involves the identification of entity types that are specific to the domain. These include statutes, defined legal

terms, legal roles, and document references (Au et al., 2022; Kalamkar et al., 2022). The unique vocabulary and complexity of legal documents require that the NER models must be trained on legal documents. This helps to achieve high accuracy (Chalkidis et al., 2022). A legal NER system is the building block for constructing a meaningful KG from legal documents.

2.4.2 Relation Extraction

While NER identifies entities (nodes), relation extraction (RE) identifies the semantic relationships between them, which correspond to the edges in a KG (Carbonell et al., 2020; Ji et al., 2022). For instance, from the sentence "Acme Corp, headquartered in West Chester, acquired Beta Inc.," RE identifies relations such as headquarteredIn(Acme Corp, West Chester). This task is crucial for building the graph's structure. Work in this area includes both closed RE for predefined sets of relation types, and open information extraction (OpenIE), which extracts relations expressed with arbitrary text (Etzioni et al., 2008).

Approaches to RE have mirrored those in NER. Early systems were rule-based, using linguistic patterns to identify relations (Hearst, 1992). These were followed by supervised statistical models that used classifiers trained on annotated data or data generated through distant supervision, a technique that aligns known KG relations with text but is prone to noise (Kambhatla, 2004; Mintz et al., 2009). Modern LLMs perform very well at the RE task (S. Kumar, 2017; Wu & He, 2019). Through proper prompting an LLMs becomes a powerful tool for extracting relationships. It is capable of doing this with minimal task-specific fine-tuning (Chia et al., 2022).

There are many challenges in successful RE. These include handling ambiguity, extracting relations that span sentences, and adapting models to new

domains. In the legal context, extracting relations, such as amendments, definitions, and obligations, is critical for building a KG that accurately reflects the legal framework (Dhani et al., 2021; Tauqeer et al., 2022). The structured output from NER and RE forms the basis for the construction of a robust KG.

2.5 Consistency, Completeness, and Coherence

The evaluation of the quality of a document involves the assessment of internal integrity. The aspects of this integrity are consistency, completeness, and coherence (Umar & Lano, 2024). These concepts have some overlap. But each addresses distinct elements. These elements are necessary for ensuring that a document is understandable, unambiguous, and effective. Here are some details:

- **Consistency:** Refers to having no logical contradictions within a document or set of documents (Egyed, 2006; Guo et al., 2023; Heitmeyer et al., 1996; Nentwich, 2005; Tröls et al., 2022; Yang et al., 2024; Zowghi & Gervasi, 2003). A consistent document must not include the assertion of mutually exclusive facts. A document must not identify different outcomes under identical conditions. Detecting any of these inconsistencies is vital for legal certainty and predictability (Donelson, 2019; Duck-Mayr, 2022; Rossi, 2016). A relevant example is that formal logic and automated reasoning techniques have been employed to verify the consistency of formal specifications (Brucker & Wolff, 2019; Heitmeyer et al., 1996).

- **Completeness:** Is the idea that a document contains all of the necessary information relative to its intended scope (Zowghi & Gervasi,

2003). Defining completeness depends on a clear specification of what should be included. Creating these specifications is a challenge. For a legal document, completeness requires that all terms be defined. It also requires that all procedures be specified and that relevant scenarios are addressed. Gaps or omissions lead to ambiguity and disputes. Assessing completeness requires significant domain knowledge. In addition, it involves the comparison of a document to its specifications (Umar & Lano, 2024; Zowghi & Gervasi, 2003). In KGs, the interpretation of completeness is further affected by the ”Closed World Assumption” versus the ”Open World Assumption” (Hitzler et al., 2009; Reiter, 1978).

- **Coherence:** Is the ability to understand the content, the organization of the content, and the flow of the information presented (Shen et al., 2021; Wang & Guo, 2014). A coherent document is well-structured, uses terminology consistently, and ensures cross-references are accurate. While related to consistency, coherence emphasizes clarity and comprehensibility for human readers. It encompasses aspects such as lexical cohesion and structure (Wang & Guo, 2014).

Ensuring all three qualities simultaneously in a large legal document is exceptionally difficult. Traditional manual review is simply not up to the task. There a a nuber of things that make this difficult:

- The volume of text
- the complex inter-dependencies
- the potential for ambiguity in natural language
- distributed authorship

(Beth, 2018). Computational approaches that leverage structured representations, such as KGs, offer a method to automate this.

A KG provides a structured framework amenable to automated analysis. It does this by creating a model of all the entities, their relationships, and attributes. Graph-based queries and algorithms can be designed to automatically identify possible inconsistencies. Some examples are conflicting property values or circular definition chains (Brucker & Wolff, 2019; Schönberg et al., 2011; Tauqeer et al., 2022; Weitl & Freitag, 2006). Perfect verification of completeness is often intractable for documents. KGs can provide a way to find any potential gaps by analyzing the graph's structure for missing nodes, absent relationships, or orphaned sections (Knublauch & Kontokostas, 2017; Omran et al., 2020; Rabbani et al., 2022; Umar & Lano, 2024).

LLMs can contribute throughout this process by interpreting text and using the results to populate the KG. They can also be used to analyze the initial KG to build a semantic hierarchy. They can formulate graph queries that will identify issues. Lastly, they can summarize the findings for human review (Benjira et al., 2025). The KG itself serves as a persistent, globally coherent, and computationally tractable structure that can be used for systematic integrity checks. Such a structure can overcome the context window limitations and the potential lack of deterministic reasoning inherent in LLMs alone. Research on KGs and related AI techniques for automated integrity checking provides the basis for this approach (Aumiller et al., 2021; Dhani et al., 2021; Heitmeyer et al., 1996; Tauqeer et al., 2022; Umar & Lano, 2024). This praxis aims to build on such work by investigating the practical application of LLM-driven KG construction for analyzing municipal legal codes.

2.6 Challenges in Analyzing Large Documents

Research in automated document processing is extensive. It encompasses many tasks. Some of the most important are summarization, information extraction, and question answering (D. Chen et al., 2017; Gambhir & Gupta, 2017; Zhong et al., 2024). In the past, this research has focused on small documents. This is because small documents do not require as much computation. They are also more practical for the human creation of a ground truth.

Many real-world applications involve much larger documents. Examples include legal contracts, technical manuals, and township ordinances. Analyzing these documents presents significant challenges, including:

- **Computational Resources:** Processing large amounts of text requires extensive memory, storage, and processing power. Computational complexity often scales nonlinearly with document length. This makes naive processing of large documents infeasible (Vaswani et al., 2017).
- **Long-Range Dependencies:** Comprehension requires capturing connections that are far apart in the document. Models with limited context windows struggle to accurately capture these relationships (Liu et al., 2025; Zhao et al., 2023).
- **Context Fragmentation:** A common technique for handling large documents involves splitting them into smaller chunks. This is necessary for models with fixed inputs. But this method risks losing context that spans chunk boundaries. This will create a fragmented understanding of the document (T. Chen et al., 2024; Qu et al., 2024).
- **Evaluation Complexity:** The assessment of the quality of a large doc-

ument is inherently difficult for human evaluators. It is almost impossible to create a ground truth for evaluation benchmarks related to large documents (Shaham et al., 2022).

The retrieval-augmented generation (RAG) technique allows LLMs to leverage information from large external sets of documents without the need to fit them into the LLMs context window (Lewis et al., 2020). RAG retrieves fragments of text from documents. They are available to the LLM as context. RAG is powerful for tasks that require deep knowledge. Standard RAG retrieves discrete, localized chunks and does not provide the holistic, structured view of the entire document. Unlike a pre-constructed KG.

2.7 Challenges in Analyzing Legal Documents

Legal documents have all the attributes that make analysis difficult. They are so important to get right that it makes the search for a solution crucial. These documents possess several characteristics that make them both difficult to work with and valuable targets for automation. Some of these are:

- **Complexity and Precision:** Legal language consists of specialized terminology and complex sentence structures. Ambiguity must be minimized. This demands high precision in interpretation. Any misinterpretations will have significant real-world consequences (Ashley, 2017; Malik et al., 2022).
- **Volume and Interconnectedness:** Legal documents are very large and highly interconnected. There are citations, amendments, and definitions. To understand just one part of a document, one must possess the ability to comprehend its relationship to numerous other parts (Beth, 2018)(Beth, 2018).

- **Semi-structured Format:** Legal documents combine structured elements (e.g., sections, clauses) with unstructured natural language. This requires powerful NLP techniques that can handle both.
- **Critical Need for Integrity:** Consistency, completeness, and coherence are very important for legal documents. These qualities are key to the rule of law. Having them ensures predictability and enforceability. Flaws will lead to disputes, costly litigation, and erosion of public trust (Donelson, 2019; Duck-Mayr, 2022; Rossi, 2016).

The use of Pennsylvania township ordinances provides a robust dataset. These codes exhibit realistic complexity. They have been developed over decades by multiple authors and through numerous amendments. The legislative process of drafting new ordinances introduces complexity. The codification process of integrating new ordinances into existing ordinances introduces complexity. While they are designed to ensure quality. This multistage human review process can still introduce errors. Once introduced, inconsistencies, incompleteness, and incoherence can persist as the code grows (Rossi, 2016). The resource-intensive and fallible nature of a purely manual review motivates the search for an automated alternative. The incorporation of computational methods to help legal professionals maintain the integrity of these legal documents is necessary.

2.8 Related Work

Research that is relevant to this praxis covers multiple topics. The application of LLMs for information extraction. The use of KGs for document integrity checking. NLP in the analysis of legal text.

LLMs for Information Extraction and KG Construction: The advent of

powerful LLMs has advanced information extraction. Studies demonstrate that models such as GPT-4, often via prompting, can perform NER and RE with performance comparable to or exceeding that of traditional fine-tuned models, particularly in specialized domains (Xu et al., 2024). Researchers have investigated many techniques to mitigate the limitations of LLMs and to control the output structure (S. Wang et al., 2023). Several works focus on constructing KGs from text using LLMs as the primary extraction engine, developing pipelines that integrate entity identification, relation extraction, and schema mapping, sometimes involving human-in-the-loop refinement (Benjira et al., 2025; Lairgi et al., 2024).

KGs for Document Analysis and Integrity Checking: Beyond construction, KGs provide a framework for advanced document analysis, including semantic search and complex question answering (Hogan et al., 2021; Ji et al., 2022). The use of KGs for consistency and completeness checking is directly relevant to this work. In software requirements engineering, KGs and ontologies model requirements to detect conflicts (Umar & Lano, 2024). In the Semantic Web community, technologies such as Shapes Constraint Language (SHACL) provide a standard way to validate RDF KGs against predefined constraints, effectively checking aspects of data integrity (Knublauch & Kontokostas, 2017).

AI and NLP for Legal Document Analysis: The legal domain has been a focus of AI and NLP research for decades (Ashley, 2017). Recent research applies modern NLP to tasks such as legal information retrieval, case outcome prediction, and contract review (Aletras et al., 2016; Chalkidis et al., 2022; Moens, 2001). Information extraction from legal texts has received particularly significant attention, focusing on extracting citations, legal entities, and obligations (Kalamkar et al., 2022; Tauqeer et al., 2022). Some

prior work has explored automated consistency checking in legal documents, often using rule- or logic-based approaches. However, these efforts have typically focused on specific conflict types rather than a comprehensive KG-based approach applied to municipal codes (Rossi, 2016).

2.8.1 Unique Linguistic Features of Legal Text

The analysis of legal documents constitutes a subfield of NLP. It can be referred to as legal AI. This is due to the unique linguistic structure, ”legalese”, employed in these documents (Ashley, 2017). This structure presents formidable obstacles for standard NLP models. They are generally trained on general-domain corpora. Key features of legal text include:

- **Deontic Modality:** Legal texts are saturated with language expressing obligations, permissions, and prohibitions, conveyed through modal verbs such as shall, must, may, and is prohibited (Navarro & Rodríguez, 2014). Accurately parsing these expressions to extract the agent (who is obligated), the action (what must be done), and the specific modal force is essential for modeling the regulatory framework encoded in the text.

- **Intertextuality:** Legal documents rarely exist in isolation. Their meaning is constructed through a dense network of internal and external citations, such as “as defined in Section 3.B” or “pursuant to 53 Pa.C.S.A. § 101.” This high degree of intertextuality distributes a document’s semantic content across multiple locations, making the modeling of these referential links an inherently graph-like problem, which is critical for coherent analysis.

- **Controlled Vocabulary and Ambiguity:** While legal language aims

for precision through a controlled vocabulary of defined terms, it remains susceptible to ambiguity. Syntactic ambiguity can arise from complex sentence structures with long, nested clauses, complicating the determination of modifier scope (the ”dangling else” problem). Semantic ambiguity can occur when a term has a specific legal meaning that differs from its common usage.

These unique challenges necessitate legally focused NLP approaches capable of moving beyond simple entity recognition to a more nuanced extraction of structured legal rules, definitions, and deontic relationships.

2.8.2 A Brief History of Computational Law

The application of computational methods to legal text is not new. The field of computational law has evolved over several decades. It reflects broader trends in artificial intelligence. Knowledge of this history provides context for the contemporary approaches leveraged in this praxis.

Early legal AI efforts, from the 1970s and 1980s, were dominated by rule-based expert systems (Susskind, 1986). These systems attempted to capture legal knowledge by having human experts manually encode statutes and case law into formal logic. Often, this was done in computer languages such as Prolog. While pioneering, these systems were brittle, struggled with legal ambiguity, and faced an insurmountable ”knowledge acquisition bottleneck” due to the labor-intensive nature of manual encoding.

The subsequent era saw a shift toward statistical methods and the rise of large-scale legal databases such as Westlaw and LexisNexis. During this period, NLP primarily focused on legal information retrieval (LIR). This employed techniques from information retrieval to improve the search and discovery of relevant documents for legal professionals. Machine learning

was applied to tasks such as e-discovery. It used algorithms to classify documents as relevant or non-relevant to a case. This drastically reduced the burden of manual review.

The modern era of computational law is defined by the use of deep learning. Most recently through the use of LLMs. Transformer-based models have achieved near human like results on various legal tasks. This includes predicting judicial decisions from case texts (Aletras et al., 2016), automated summarization of legal opinions, and contract analysis and clause extraction. The research presented in this praxis operates within this modern paradigm. The application of the powerful generative and analytical capabilities of LLMs. They are used to address the foundational yet often overlooked task of ensuring the internal structural integrity of legislative documents.

Positioning of This Work: This praxis builds on these converging lines of research. While previous work has explored LLMs for KG construction and KGs for consistency checking independently. The specific contribution of this work lies in the integration and practical application of modern LLMs to construct KGs from municipal legal codes for consistency and completeness analysis. The project addresses the limitations of the LLM context window by leveraging the KG structure for global reasoning. Unlike prior legal AI research, which focused on case law or contracts. This project targets foundational legislative texts at the local government level. The ”praxis” aspect emphasizes the development and evaluation of a practical methodology to assist municipal professionals in maintaining the quality of their codified laws.

2.9 Conclusions

This literature review highlights a foundational tension in modern NLP. While transformer-based LLMs possess unprecedented text processing capabilities. Their inherent context window limitations prevent the holistic analysis required for verifying the integrity of large documents (Liu et al., 2025; Vaswani et al., 2017). The research surveyed demonstrates that KGs provide a robust solution by transforming unstructured text into a persistent, structured, and globally queryable representation of knowledge (Hogan et al., 2021).

The process of information extraction, encompassing named-entity recognition (NER) and relation extraction (RE), serves as the critical bridge between unstructured documents and the structured KG (Xu et al., 2024). Through a KG, the core quality attributes of a document (consistency, completeness, and coherence) become computationally tractable. This allows for a systematic analysis capable of overcoming the limitations of manual review. Which is particularly error-prone in dense, interconnected legal corpora (Beth, 2018; Zowghi & Gervasi, 2003).

The following chapter details the specific methodology employed for constructing and evaluating a system designed to achieve the objectives of this praxis.

Chapter 3: Methodology

3.1 Introduction

This chapter describes the methodology used to develop a system that is based on large language models (LLMs). It uses the LLM to analyze a document and turn it into an attributed Knowledge Graph (KG) (Fensel et al., 2020; Hogan et al., 2021). As established in Chapter 1, these KGs serve as the data structure for the analysis of document completeness and consistency (Umar & Lano, 2024; Zowghi & Gervasi, 2003). The challenge of understanding the meaning of large documents requires automated solutions like this (Bhattacharya et al., 2019). Traditional manual review is insufficient for ensuring comprehensive consistency in large documents. This research is based on the idea that KGs offer a viable approach to addressing these challenges (Zhong et al., 2024).

The methodology presented here outlines the overall research approach. First, the system architecture is presented. Next, the data sources are described. Pennsylvania township laws are the data used in this research. The chapter then details the hyperparameters that govern the system. Then, the experiments planned for hyperparameter tuning are specified. Details of the software tools utilized are then presented. Then, the specific strategy for evaluating the quality of the generated KGs is presented. Since we do not have a ground truth for every case, this issue is addressed. Finally, the inherent limitations of the chosen methods and the relevant ethical considerations are examined. It concludes with a restatement of the research questions and hypotheses that this research will address.

3.2 Approach

This research introduces Mnemosyne, a multi stage pipeline that transforms large documents into KGs. These KGs can be queried to understand issues with the content of the source document. Hereafter, this system will be referred to as Mnemosyne. By leveraging LLMs (Benjira et al., 2025; Lairgi et al., 2024), the system is designed to facilitate the automated analysis of documents for internal consistency and completeness. The architecture prioritizes two critical features. It requires minimal specialized skills during operation. Secondly, every piece of information in the KG can be linked to its source. While developed using legal documents, the modular design is generalizable to other domains (Shaham et al., 2022).

3.2.1 Architectural Overview

The system employs a modular pipeline that is based on established KG construction practices (Ji et al., 2022). It includes a two tiered memory architecture. This design manages the context window limitations of modern LLMs (Liu et al., 2025). The KG is built in iterations. The data flows through a sequence of components, from initial document processing to advanced KG enrichment. This modularity permits targeted optimization at each stage. Figure 3.1 provides a high-level depiction of this data flow. Note that processes P1, P3, P4, and P5 are covered in detail below. Processes P2 and P6 are out of scope and not covered, but they are included for future reference.

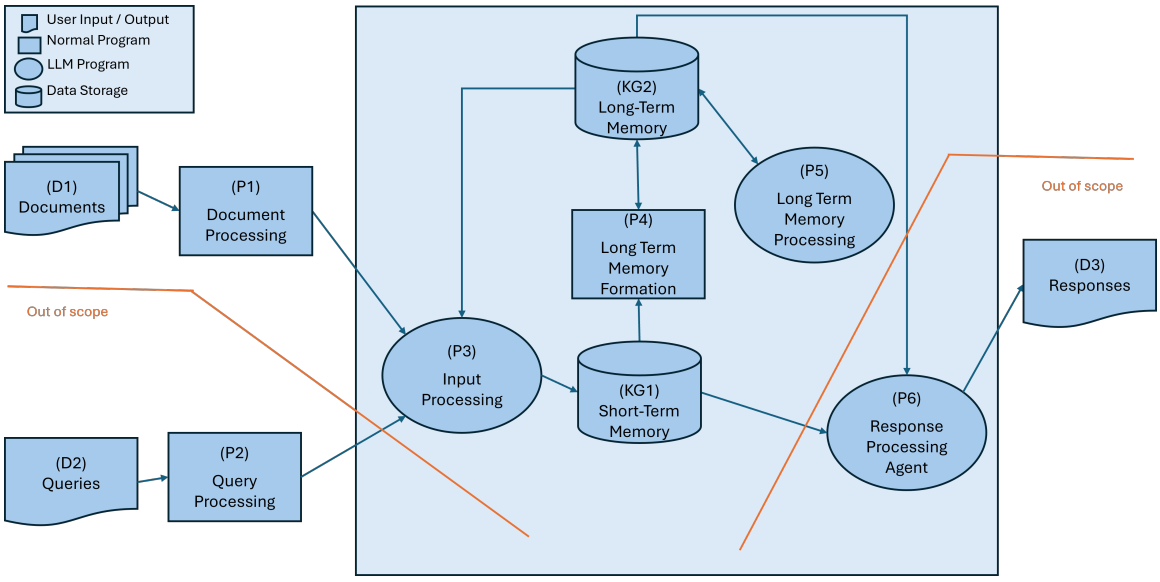


Figure 3.1: *High-Level Overview of the Document-to-KG Pipeline*

3.2.1.1 Document Ingestion and Chunking (P1)

The pipeline starts with the ingestion of one or more source documents. The code was built to handle multiple documents, and early trials did build KGs from multiple documents. But for the remainder of the work, only single documents were processed. This simplified tracking and reporting. is the start of the pipeline.

The documents are segmented into units called *chunks*. This chunking process is a critical step in overcoming the fixed context window of LLMs. A naive split can sever key semantic links. Therefore, this component implements and evaluates several intelligent strategies. They include fixed size token windows, paragraph-based segmentation, and overlapping chunks. Each chunk is meticulously annotated with location metadata (e.g., page number, section number) to guaranty traceability.

3.2.1.2 Justification for Paragraph-Based Chunking (P1)

It is critical to have a good strategy to derive chunks from the source document. The strategy directly impacts the semantic integrity of the data fed to the LLM. While methods like fixed size token windows offer computational efficiency. They are fundamentally agnostic to the document's underlying logical structure. Arbitrarily splitting a sentence or paragraph can lead to catastrophic context loss, a risk that is unacceptable.

Paragraph set-based chunking offers an improvement and balances the processing efficiency of small sets of words, tokens, or sentences with the improved context of sets of paragraphs. It also removes the computational complexity of identifying semantic boundaries. However, it can result in information loss at chunk boundaries. For this reason, Mnemosyne adopts the overlapping paragraph set for chunking. For example, the paragraph set could be configured to consist of 20 paragraphs with an overlap of five paragraphs. In this case, paragraphs 1 - 20 would be the first set, 16 - 35 would be the second set, 31 - 50 would be the third set, and so on.

This **paragraph-based strategy** is justified by its focus on preserving the most basic, self-contained unit of meaning. Each paragraph set is treated as a discrete chunk, ensuring that a complete thought is never fragmented. Headings are processed no differently than body text. They are simply treated as another paragraph within the document sequence. This approach does not take into account the structure of the document. This is by intention. With this approach, the system does not depend on well-structured source documents. The burden of determining the meaning of the heading and the text is delegated to the LLM.

3.2.1.3 LLM-Based Information Extraction (P3)

This component is the information extraction engine. For each text chunk, an engineered prompt instructs an LLM to perform both Named Entity Recognition (NER) and Relation Extraction (RE) in one step. The model identifies entities corresponding to a predefined legal ontology (e.g., `Ordinance`) and their semantic relationships (e.g., `AMENDS`). To ensure structured and reliable output. The LLM's generation is constrained to a predefined JSON schema (see Appendix D). The result for each chunk is a small KG based on the chunk. This design allows for parallel processing of chunks for enhanced throughput. The prompt used can be found in Section E.1.

3.2.1.4 Short-Term Memory (STM): Batch Aggregation (KG1)

As KG fragments are generated, they are placed in the Short Term Memory (STM). This is an in memory buffer. The STM functions as an efficient staging area. It accumulates KGs until a predefined batch size is reached. This batching strategy optimizes performance. It reduces the I/O overhead associated with numerous small writes to Long Term Memory (LTM). It creates an opportunity for consolidation before committing data to long-term storage. See Section C.1.1.

3.2.1.5 LTM Ingestion and Entity Resolution (P4 & KG2)

Once the STM reaches capacity, the LTM Ingestion process transfers the batch of KG fragments into the persistent Neo4j graph database. A crucial step in this process is *entity resolution*, which de-duplicates nodes. For entities with explicit identifiers (e.g., an ordinance number), the resolution is a straightforward merge. For entities without unique IDs, the system employs

a hybrid approach. First, nodes are merged based on high a similarity (e.g., Levenshtein distance). In a subsequent refinement stage, this is augmented by having the LLM identify the similarities of nodes. This ensures that identical concepts are represented by a single canonical node. See Section C.1.2.

3.2.1.6 Asynchronous KG Refinement and Enrichment (P5)

The final stage is an asynchronous engine that performs refinement operations to enhance the global KG's semantic richness. This process operates iteratively on small, random subsets of the LTM KG. It ensures that the KG will gradually improve without blocking ingestion. Key operations include:

- **Ontological Classification:** Formalizing type hierarchies by asserting `ISA` relationships (e.g., "Ordinance_123" `ISA` "Ordinance") (Minsky, 1974; Noy & McGuinness, 2001).
- **Meronymic Relationship Modeling:** Establishing the document's structure by adding part whole (`PARTOF`) relationships (e.g., "Section_3.B" `PARTOF` "Article_III").
- **Type System Refinement:** Merging semantically equivalent entity types (e.g., consolidating "Regulation" and "Rule") through the use of an LLM (Gardazi et al., 2025).
- **Ontology Organization:** Structuring entity types into a coherent subsumption hierarchy using LLM-based classification (Tian et al., 2022).
- **Instance Type Correction:** Re-classifying entity instances based on new evidence within the evolving ontology.

This two-tiered memory architecture effectively decouples initial data aggregation from deeper semantic enrichment, creating a robust and scalable

framework.

3.2.1.7 Prompt Engineering Strategy

The interaction with the LLM is entirely through prompts. This makes their design a key component of the methodology. The approach is centered on a principle of **structured, few-shot prompting** to maximize reliability and accuracy. A prompt in the system consists of four key components:

1. **Role and Goal Definition:** The prompt begins by assigning a role to the LLM (e.g., "You are an expert legal analyst...") and clearly stating the objective. This primes the model for the specific task domain.

2. **Schema and Type Definitions:** The prompt explicitly provides the complete JSON schema for the output and concise definitions for each entity and relationship type. This reduces ambiguity and helps the model correctly classify the extracted information. In early iterations, the LLM would often generate output that was different from the required JSON. Over time, the prompt was enhanced to make the JSON requirement more strict. By the time the core experiments were being run, the LLM was only generating non compliant JSON once every 15 or so LLM calls.

3. **Few-Shot Examples:** Several canonical examples of input text and the corresponding correct JSON output are included. This in-context learning is vital for guiding the model's output to match the desired structure. As described above, these examples needed to be expanded to gain better compliance from the LLM.

4. **Input Text and Final Instruction:** Finally, the text chunk to be processed is provided, followed by a concluding instruction that reiterates

the primary command.

This multi-part prompt structure is designed to be robust. It constrains the LLM's generative capabilities towards a deterministic and accurate extraction task. Content of the critical Mnemosyne prompts can be found in Appendix E.

3.3 Document Processing Workflow

The journey of a single document through the system follows the pipeline to incrementally build and refine the KG. This workflow is illustrated in the sequence of diagrams from Figure 3.2 to Figure 3.5.

When the process begins, an incoming document is converted into a sequence of chunks based on the chunk size and chunk overlap hyperparameters. As shown in Figure 3.2, the first chunk is passed to the Input Processor. The Input Processor analyzes the text to generate a KG fragment. This initial fragment is then stored in the Short Term Memory (STM), which was empty prior to this operation.

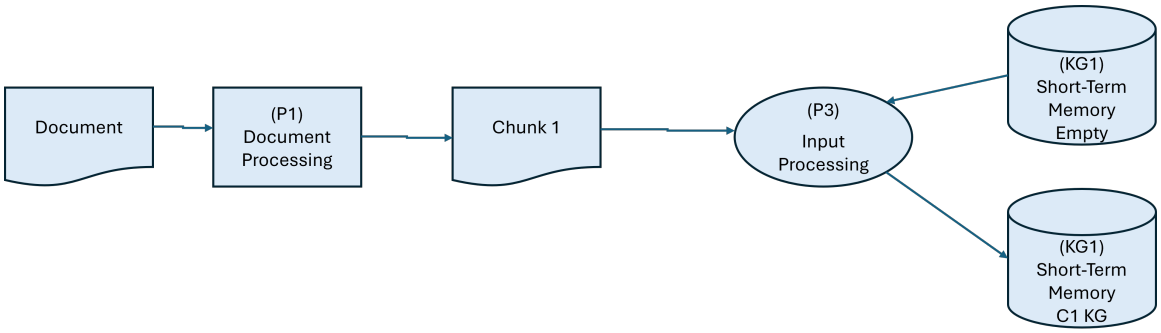


Figure 3.2: *Chunk 1 Processing*

Following the processing of the first chunk, the system proceeds to the next one. Figure 3.3 depicts this step. The Input Processor analyzes "Chunk 2" and generates a new KG fragment. The new KG fragment is then added

to the STM. At this stage, the STM contains the KG fragments from both the first and second chunks.

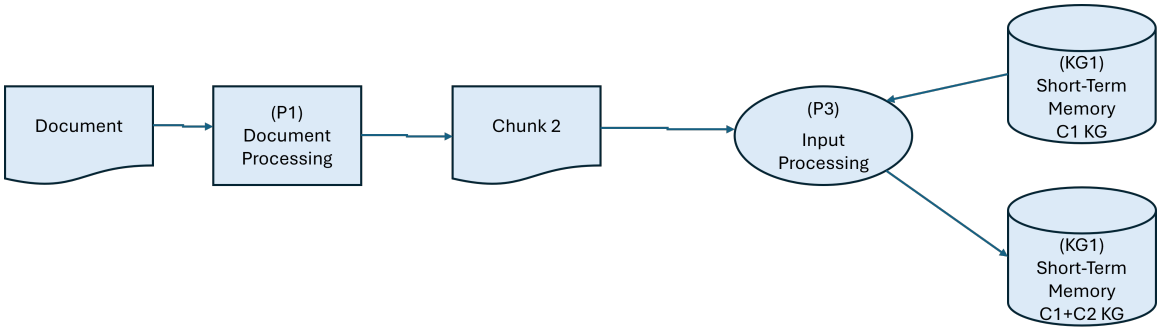


Figure 3.3: *Chunk 2 Processing*

This process continues until the STM reaches capacity specified in the maximum chunks hyperparameter. At this point, the LTM Formation process is triggered. This is illustrated in Figure 3.4. The entire collection of KG fragments held in the STM is transferred for consolidation. This phase merges the individual fragments, resolves redundancies, and integrates the new information into the persistent LTM. The STM is then cleared, ready for the next batch. During this process, any other use of the STM is clocked.

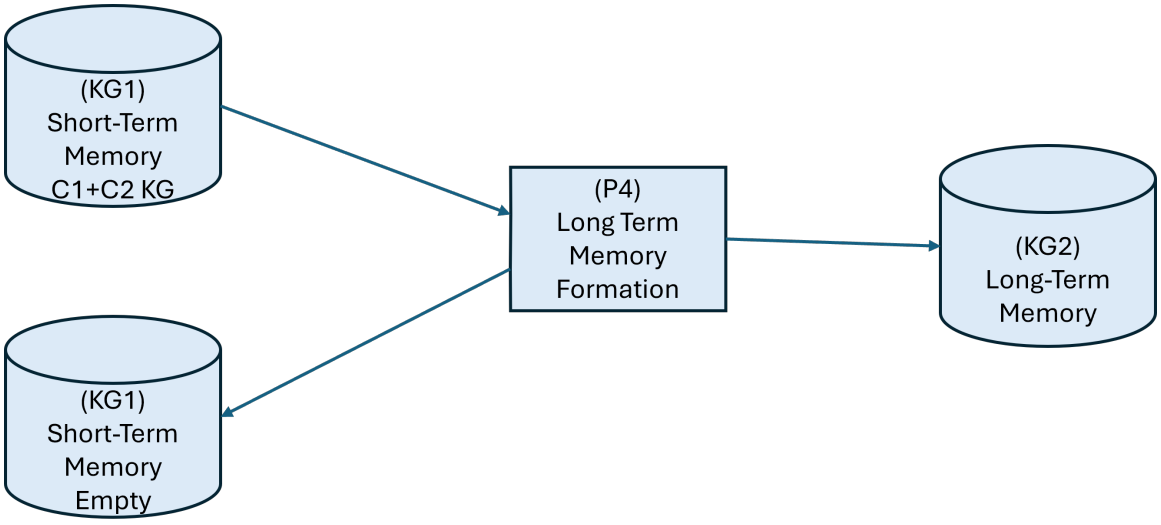


Figure 3.4: *Full Short Term Memory Processing*

Finally, as depicted in Figure 3.5, an asynchronous LTM Processing engine operates in the background. It performs computationally intensive refinement tasks on the global KG. This continuous refinement ensures that the knowledge base becomes more accurate, coherent, and semantically rich over time. This dual memory architecture is key to balancing the demands of rapid processing with the need for deep semantic integration.

The code to support this asynchronous background processing was developed and tested. Some early runs made use of it. However, for the analysis of timing and direct control over the number of refinement steps, it was not used during the experiments.

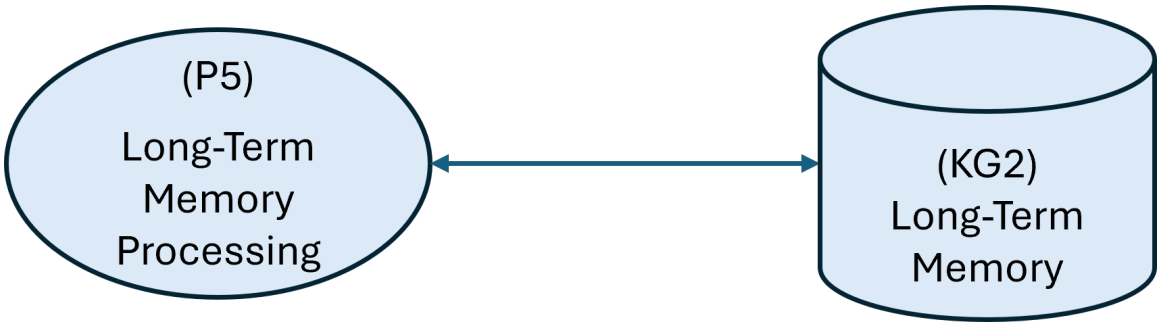


Figure 3.5: *Long-Term Memory Processing*

3.4 Alternative Methodologies Considered

Several alternatives were considered in developing the approach described here. The approach selected was based on research and preliminary tests. These alternatives are discussed below.

3.4.1 KG vs. Retrieval-Augmented Generation (RAG)

The first alternative approach that was considered was a pure Retrieval Augmented Generation (RAG) system. In a RAG model, a document is

chunked and stored in a vector database. When a query about consistency is posed, the system retrieves relevant chunks and feeds them to an LLM. While this can be very effective for question answering, RAG is ill suited for verifying global consistency. Detecting a conflict might require addressing information from two chunks that are not near each other (e.g., a definition in Chapter 1 and a conflicting application in Chapter 20). With the RAG approach, retrieval is 'on demand' and localized. Whereas the KG based methodology creates a persistent, holistic model of the entire document. This allows for graph wide structural queries. These queries can uncover long distance dependencies and inconsistencies.

3.4.2 Property Graph vs. RDF/SPARQL

Also considered was the use of the RDF triple store model with SPARQL. But this feels limiting in terms of the amount of information that can be stored about a node or a relationship. The property graph model used by Neo4j was selected for its flexibility and intuitive query language (Cypher). Property graphs allow properties to be stored on both nodes and edges. This is useful for attaching metadata such as 'effectiveDate' to an 'AMENDS' relationship. Cypher's ability to match complex patterns is very useful for this research.

3.5 Knowledge Graph Data Model for Legal Text Analysis

A robust data model is key to the development of a KG capable of representing legal documents. This section details the proposed ontological framework for the Neo4j based LTM. It is engineered to represent both legal concepts and the efficiency of analytics.

3.5.1 Ontological Framework: Node and Edge Schema

The core entities within legal documents are modeled as nodes with labels and properties, as shown in Table 3.1. The semantic connections between these nodes are represented by directed, typed edges (relationships) detailed in Table 3.2.

Node Label	Description	Key Properties
`Section`	A structural component of the legal code, such as a chapter, article, or section.	`id`, `title`, `textContent`, `sourceLocation`
`DefinedTerm`	An explicit definition of a specialized term provided within the legal text.	`term`, `definitionText`, `sourceLocation`
`Ordinance`	A formal legislative act, such as a law or an amendment.	`ordinanceNumber`, `enactmentDate`, `title`
`ZoningDistrict`	A geographically delineated area subject to a specific set of land use regulations.	`districtName`, `districtCode`
`PermittedUse`	A specific land use that is legally sanctioned within a given zoning district.	`useName`, `conditions`
`Obligation`	A deontic expression imposing a legal duty or permission on a specific actor.	`actor`, `action`, `modality` (e.g., MUST, MAY)
`Reference`	A citation pointing to another legal provision.	`targetIdentifier`, `referenceType`, `sourceText`

Table 3.1: *Node Schema and Property Definitions*

3.5.2 Model Application: Inconsistency Detection

This ontological structure provides the foundation for the support of graph queries that may identify potential inconsistencies. A critical validation is

Relationship Type	Description (Source → Target)	Properties
CONTAINS	DocumentSection → DocumentSection. Models the hierarchical structure of the code.	–
DEFINES	DocumentSection → DefinedTerm. Links a section to a term it formally defines.	–
AMENDS	Ordinance → DocumentSection. Signifies that a law modifies a part of the code.	date
USES_TERM	DocumentSection → DefinedTerm. Denotes the usage of a defined term.	–

Table 3.2: *Relationship Schema and Semantic Descriptions*

to detect terms that are used but never formally defined. Such a scenario can be identified declaratively using a graph pattern matching language like Cypher. The query in Figure 3.6 demonstrates how multiple types of structural incompleteness such as undefined terms, unused definitions, and missing inverse relationships can be systematically identified. The ability to execute such queries demonstrates the analytical utility of the proposed data model for auditing the logical coherence of the legal code.

```
% Test 1: Find terms that are used but not defined
MATCH (a)-[:USED_IN]->(b)
WHERE NOT EXISTS (()-[:DEFINES]-&gt;(b))
RETURN
a.id AS sourceNode, b.id AS targetNode,
'Undefined Term Used' AS incompletenessType
UNION ALL
% Test 2: Find terms that are defined but never used
MATCH (a)<-[:DEFINES]-(b)
WHERE NOT EXISTS (()-[:USED_IN]-&gt;(a))
RETURN
a.id AS sourceNode, b.id AS targetNode,
'Unused Definition' AS incompletenessType
UNION ALL
% Test 3: Find AMENDS relationships missing the inverse AMENDED_BY
MATCH (a)-[:AMENDS]->(b)
WHERE NOT (b)-[:AMENDED_BY]->(a)
RETURN
a.id AS sourceNode, b.id AS targetNode,
'Missing AMENDED_BY' AS incompletenessType
```

Figure 3.6: *Cypher query to detect multiple types of incompleteness*

3.6 Dataset Selection and Rationale

The primary dataset comprises the codified ordinances of various townships within the Commonwealth of Pennsylvania. This choice is predicated on several factors: these documents are voluminous and linguistically complex; their public availability supports open research; they contain rich interrelationships that are well suited for KG representation; and they directly address the problem of inconsistency in municipal laws identified in Chapter 1 (Curley, 2024; Rau, 2024).

3.7 Data Acquisition and Pre-processing

The documents were acquired from online public sources, primarily the legal code aggregator eCode360 (Sanders, 2024), in DOCX format. This format is preferred over PDF because it preserves structural information advantageous for semantically aware chunking. An initial Exploratory Data Analysis (EDA) on a sample of municipal codes was conducted to characterize typical document structure, identify common entity and relationship patterns, and analyze linguistic features. The findings from the EDA informed the refinement of the KG data model and LLM prompts. The preprocessing pipeline involves extracting core textual content, removing irrelevant artifacts (e.g., headers, footers), and normalizing the text to create a clean corpus.

3.8 Hyperparameters

The pipeline's performance and operation are governed by hyperparameters that must be tuned. They were tuned on smaller documents so that many

cycles could be run. Some parameters were copied directly from the tuning runs. Some were scaled up from the tuning runs. For example, the tuning documents were so small that the chunk size had to be kept small. But it was still varied. And the learning from this led to the large chunk size used in the main experiments. These parameters are summarized in Table 3.3.

Pne of the most important hyperparamerts to selct is the choice is the LLM Model. Models from Google Gemini, OpenAI GPT, and Anthropic Claude families were evaluated. Criteria included their extraction accuracy, adherence to structured formats, and cost. The LLM Temperature was set low (0.0 to 0.1) to favor deterministic, factual extraction and to minimize hallucinations (Rzepka et al., 2023). Prompt Design was an iterative process to improve the clarity and effectiveness of few-shot examples.

Document chunking strategies are vital for managing LLM context windows (Ratner et al., 2022). Different strategies were tested, including varying the number of paragraphs per chunk and the overlap percentage to balance context continuity with redundant processing (Qu et al., 2024).

Finally, there are processing parameters that control the KG refinement pricess. The STM Fullness Threshold dictates the batch size for LTM ingestion. The LTM Formation and Processing Batch Sizes control the resource intensity and depth of the consolidation and refinement cycles.

Category	**Parameter**
LLM Used	LLM Model, LLM Temperature, Prompt Design
Document Chunking	Strategy, Paragraphs per Chunk, Overlap Percentage, Chunk Size (Tokens)
Processing	STM Fullness Threshold (Number of KGs), LTM Formation Batch Size, LTM Processing Batch Size (Number of Nodes), Frequency of LTM Processing

Table 3.3: *Hyperparameter Categories and Specific Parameters*

3.8.1 Hyperparameter Tuning Methodology

Mnemosyne has a large number of tunable hyperparameters. Therefore, an exhaustive grid search is not feasible. Therefore, an alternative approach was needed. After several attempts and research, an iterative tuning approach was selected. In this process, a baseline configuration is established. Then, a single hyperparameter is varied at a time. The tuning proceeded in the following sequence:

1. **LLM and Prompt Selection:** First, the LLM model was selected based on a balance of F1 score, cost, and processing time. This was done on a representative data sample. The prompt was refined to ensure it was working as expected.

2. **Chunking Strategy Optimization:** With the selected LLM model, the next step is to determine the chunk parameters. The focus was on the size and overlap. The goal was to maximize the context provided to the LLM without exceeding its limits or incurring excessive overhead.

3. **Processing and Refinement Tuning:** Finally, parameters governing the STM and LTM processes (e.g., batch sizes, fullness thresholds) were tuned. This was to optimize the system throughput and stability.

This pragmatic methodology established the final values used for the main experiments described in Chapter 4.

3.9 System Implementation and Environment

The methodology was implemented using a standard set of Machine Learning and AI tools. They are:

- **Core Language:** Python (version 3.12.12) for its extensive data science and NLP libraries.
- **Development Environment:** Google Colaboratory for access to computational resources, including GPUs.
- **Graph Database:** Neo4j (version 2025.10) is a native graph database system. It uses the Cypher query language to navigate the graph (N. Kumar & Kumar, 2013).
- **Document Parsing:** The ‘python-docx‘ library is used to extract textual content and structural information.
- **Version Control:** Git and GitHub to manage all code and configuration files. This ensures version control and that experiments are reproducible.

3.10 Evaluation Measurements

Evaluating the quality of the generated KG is not trivial because it is difficult to establish a ground truth for large documents (Dhani et al., 2021). Therefore, the evaluation strategy is to use multiple approaches. The quantitative, qualitative, intrinsic, and extrinsic measures are combined as summarized in Table 3.4.

3.10.1 Knowledge Graph Fit Assessment

This evaluation is focused on how well the generated KG represents the source documents.

- **Entity/Relation Extraction Quality:** The core extraction quality is measured using Precision, Recall, and F1 score. This is run using a

set of small documents that have manually created ground truth. It is also run against the small documents and the large documents using ground truth generated by Google's LangExtract tool.

- **Traceability:** This is measured by sampling nodes and relationships to verify that their 'source_location' property correctly links back to the text in the source document. The target for this metric is over 95% accuracy.

- **KG Structural Integrity:** This is analyzed by examining graph level statistics (e.g., node/edge distributions, density) before and after LTM Processing. The goal is to show that the refinement process leads to a better graph structure.

3.10.2 Consistency and Completeness Analysis

This assessment determines whether the KG can be used to check Consistency and Completeness (C&C).

- **Presence of Requisite Information Types:** A checklist-based evaluation ensures that the information types necessary for C&C checks (e.g., definitions, obligations, and cross references) are successfully extracted.

- **Queryability for Integrity Patterns:** A set of predefined Cypher queries is executed to test for common integrity patterns, including terms that are used but never defined or sections that are referenced but not present.

- **KG Schema/Ontology Quality:** The ontology of entity types that emerges is qualitatively assessed for its clarity, coherence, and appropriateness to the legal domain.

3.10.3 Controlled Error Injection Experiment

To provide an objective measure of the system's ability to facilitate error detection, a controlled experiment was conducted. A "gold standard" document is seeded with common errors to create a synthetic ground truth (Mintz et al., 2009). The typology of injected errors includes:

- **Type 1:** Contradictory Definition
- **Type 2:** Undefined Term Usage
- **Type 3:** Broken Cross Reference
- **Type 4:** Conflicting Obligation
- **Type 5:** Incomplete Specification

The primary metric is the **Error Detection Rate (EDR)**, the percentage of injected errors identifiable through specific KG queries.

Category	Metric
Knowledge Graph Fit	Entity/Relation Extraction (Precision, Recall, F1 score), Content Coverage & Plausibility, Traceability, KG Structural Integrity
Suitability for C&C Analysis	Presence of Requisite Information Types, Queryability for Integrity Patterns, KG Schema/Ontology Quality
Error Injection	Error Detection Rate (EDR), Traceability of Error Indicators

Table 3.4: *Summary of Measurement Metrics*

3.10.3.1 Quantitative Extraction Metrics

To formally measure extraction performance, the standard metrics of Precision (P), Recall (R), and F1 score are used, calculated from counts of True

Positives (TP), False Positives (FP), and False Negatives (FN).

$$\text{Precision (P)} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.1}$$

$$\text{Recall (R)} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.2}$$

$$\text{F1-score} = 2 \times \frac{\text{P} \times \text{R}}{\text{P} + \text{R}} \tag{3.3}$$

The F1 score provides a single metric that balances Precision (accuracy of extractions) and Recall (completeness of extractions).

3.11 Methodological Limitations

There are multiple limitations to this research. The system's accuracy is dependent on the capabilities of the underlying LLM, which can hallucinate or misinterpret nuanced text (Zhao et al., 2023). Document chunking risks losing context that spans across chunks (Qu et al., 2024). While this is mitigated with overlaps, it is still possible. KG evaluation involves subjective judgment in the absence of a comprehensive ground truth. Note that errors early in the pipeline stages can propagate (Zhong et al., 2024). The scope is limited to textual content, and the system is evaluated in a specific legal domain. Performance on other document types may vary. Finally, this work focuses on generating a KG suitable for C&C checks; the implementation and evaluation of a fully automated C&C system are designated as future work (Zowghi & Gervasi, 2003).

3.12 Ethical Considerations

The research is guided by several ethical principles. The data consists of public records, and no private or sensitive information is being processed. The system is intended as an assistive tool for legal experts, not a replacement for human judgment. The emphasis on traceability mitigates the risk of over reliance and promotes transparency. The potential for LLMs to perpetuate biases present in their training data is acknowledged. Although the narrow domain of legal texts may limit the impact of broader societal biases (Zhao et al., 2023). All intellectual property will be handled according to university policy.

3.13 Conclusion

This chapter has presented a detailed methodology for developing and evaluating an LLM based system to convert large documents into KGs. The approach specifies a modular, multi stage pipeline. It includes a comprehensive data model. There is a systematic plan for hyperparameter tuning. In addition, there is a rigorous evaluation strategy that combines quantitative metrics, qualitative reviews, and a controlled error injection experiment. By addressing inherent limitations and adhering to ethical considerations, this methodology provides a structured and defensible plan to investigate the research questions and test the hypotheses set forth in Chapter 1.

Chapter 4: Results and Analysis

4.1 Chapter Overview

In this chapter, the results from the experiments detailed in Chapter 3 are presented. They are organized to address all of the research questions and hypotheses. The results show how Mnemosyne was able to use OpenAI's GPT to construct a knowledge graph (KG) for the validation of documents. The goal of this research is validated across the three dimensions of correctness, completeness, and consistency.

The first section contains the details of the experimental setup. This setup includes:

1. the computational environment
2. the rationale for the dataset selection
3. the methodology for establishing a ground truth for benchmarking system performance.

The outcomes from the preliminary experiments that focus on hyperparameter tuning are presented. This tuning work was used to determine the proper hyperparameter values for the main experimental runs. This was done by methodically testing variables such as model selection, chunking strategy, and refinement cycles.

The main results are presented in three parts, organized to reflect the evaluation framework defined in Chapter 3:

1. **Knowledge Graph Fit Assessment:** Evaluating the quality, traceability, and structural integrity of the generated KGs.

2. **Suitability for Consistency and Completeness (C&C) Analysis:** Assessing the KG's utility for performing automated validation tasks.

3. **Controlled Error Injection Experiment:** Quantitatively analyzing the system's ability to detect seeded inconsistencies and instances of incompleteness.

Following the presentation of data, the chapter provides an in-depth analysis and interpretation of the findings. This analysis refines the initial research hypothesis. Notably, this introduces a more intricate two-part understanding of completeness. This is a key contribution of this work. Finally, each research hypothesis is validated against the collected evidence. The chapter concludes with a summary of the key results. Thereby, setting the foundation for the discussion and conclusions in Chapter 5.

4.2 Experimental Setup

To test the research hypotheses, the software tool Mnemosyne was developed. As described in Chapter 3, the tool's pipeline:

1. ingests a source document
2. divides it into manageable chunks using a paragraph set strategy
3. generates a preliminary KG for each chunk via a Large Language Model (LLM)
4. merges these individual graphs into a unified structure
5. iteratively refines the merged KG.

This results in an enriched KG with IS_A (taxonomic) and PART_OF (meronymic) relationships. Through a series of structured experiments, the efficacy of this tool and its underlying methodology was evaluated.

4.2.1 Computational Environment

The experiments were conducted using the technical environment specified in Section 3.9. This comprises Python (v3.8+) and a Neo4j (v5.x) graph database. Python was used for all processing and orchestration. The KG was used for storage and querying.

The initial development was performed using the free tier of Google Colab. As work progressed, the processing demands of the full documents increased, and the environment was upgraded to Colab Pro. This still was not enough, and so it was ultimately upgraded to Colab Pro+. This was needed to accommodate longer runtimes (up to 24 hours) required for end-to-end processing of the larger test documents. As an alternative, a local machine setup was also configured for flexibility. However, this introduced challenges regarding version control integration with GitHub. Therefore, the decision was made to rely on the stable and high-performance Colab Pro+ environment for all subsequent experimental runs. Unfortunately, this setup required active monitoring. If the connection to the remote runtime is lost when the local machine enters sleep mode, long-running experiments will be interrupted.

4.2.2 Documents

The selection of documents was critical to testing the system's capabilities across varying levels of complexity and scale. Two main categories of documents were used. Small narrative texts for controlled, rapid testing,

and large, complex legal documents for real-world application and scalability assessment.

1. **Test Documents:** Four small narrative documents (1–4 pages each) were created to facilitate rapid development and iterative testing of core extraction and refinement logic. Their limited size made manual creation of a ”gold standard” ground truth feasible. This allowed for direct quantitative measurement of extraction accuracy (precision, recall, F1 score) during the hyperparameter tuning phase. A detailed exploratory data analysis (EDA) for these narrative documents is presented in Appendix F.

2. **Small-Scale Experimental Document:** The Code of Ordinances for the Conewago Township Sewer Authority (CTSA), an 82-page document, was selected as the primary subject for the main experiments. Its size is substantial enough to present a realistic challenge for the chunking and consolidation pipeline. Yet it remains manageable for detailed analysis and multiple experimental runs. To rigorously test the system's ability to detect specific flaws, four versions of this document were prepared:

    - **Base Document:** The original, unmodified text, serving as the control against which all modifications are compared.
    - **Incompleteness Injection:** This version removed the entire section on ”Delinquency” to test a challenging form of external incompleteness, the absence of a complete, self-contained, and unreferenced topic. This was used to determine whether such an omission could be detected solely by internal structural analysis.

- **Inconsistency Injection (Dissimilar Topic):** A section on ”Driveways” from the Easttown Township code (ET) was inserted to test the system's ability to identify content semantically inconsistent with the surrounding domain (i.e., sewer regulations vs. road construction). The hypothesis was that this semantically distant content would form a distinct, poorly connected cluster in the KG. Thereby making it identifiable through graph-based topological analysis.
- **Inconsistency Injection (Similar Topic):** A section on ”Prohibited Wastes” from another township's sewer code was inserted to present a more subtle and challenging test. Could the system detect inconsistencies when the vocabulary and domain are nearly identical? This is a case where simple semantic clustering might fail to distinguish the foreign content.

3. **Large-Scale Experimental Document:** The ET, a comprehensive 732-page document, was selected to test the scalability and performance limits of the Mnemosyne pipeline on a significantly larger and more complex legal document.

4.2.3 Ground Truth Establishment

To quantitatively evaluate the KG extraction pipeline's performance, a ground truth was established. For the four smaller narrative test documents, entities and relationships were manually annotated to create a ”gold standard” reference.

Given that manual annotation is impractical for large documents, Google's LangExtract tool was employed to automatically generate a baseline ground

truth for comparison. However, a critical architectural mismatch was identified during preliminary validation. Both the manually created and LangExtract-generated ground truths consist of shallow, one-level hierarchies. Whereas the Mnemosyne system is explicitly designed to build deep, multi-level type hierarchies through iterative refinement. This fundamental difference limits the maximum achievable F1 scores in a direct comparison. As Mnemosyne is penalized for creating a richer, more abstract ontology.

4.2.4 Experiment Groups

All experiments were organized into logical groups, identified by a numerical series, to ensure systematic execution, logging, and reproducibility. The structure of these groups, from housekeeping scripts to main experimental runs and post-run analysis utilities, is outlined in Table 4.1.

Group Range	Group Name	Purpose
0 - 99	Pre Utility	Housekeeping scripts, such as clearing the database before a run.
100 - 199	Ground Truth	Generation of ground truth KGs using Google's LangExtract tool.
200 - 299	Regression Test	Core functionality tests used for regression testing during development.
300 - 399	Hyperparameter	Experiments designed to select optimal hyperparameters for the main runs.
400 - 499	Main Experiments	Full experimental runs on the primary small and large scale documents.
900 - 999	Post Utility	Post run utilities for result analysis, visualization, and reporting.

Table 4.1: *Overview of Experiment Groups*

4.3 Exploratory Data Analysis of Source Documents

An EDA was conducted on the source documents to characterize their structural and linguistic features. This analysis establishes a quantitative baseline for the corpora, highlights the unique challenges posed by dense legal text, and provides justification for the methodological choices (such as structure-aware chunking) implemented in the Mnemosyne pipeline.

4.3.1 Corpus Summary

The two primary corpora, the short narrative stories used for initial testing and the municipal codes for scaled analysis, differ dramatically in scale. As shown in Table 4.2, the legal corpus is nearly 75 times larger than the narrative corpus in word count. This vast difference underscores the necessity of a scalable processing architecture, as a system optimized for short narratives would likely fail when confronted with the sheer volume and complexity of the legal texts.

Metric	Stories (4 Docs)	Legal (2 Docs)
Word Count	4,057	303,646
Paragraph Count	94	15,210
Sentence Count	247	15,146
Unique Word Count	1,618	11,679

Table 4.2: *Aggregate metrics for the two document corpora*

4.3.2 Comparative Document Metrics

Analysis of linguistic features reveals a stark contrast between the two corpora. As detailed in Tables 4.3 and 4.4, the legal documents exhibit significantly lower lexical diversity (0.10 and 0.03) compared with the narra-

tive stories (which average 0.40). This indicates a highly specialized and repetitive vocabulary, characteristic of legal writing. Furthermore, the average sentence length of approximately 20 words in the legal codes, combined with their complex clause structures, indicates greater syntactic complexity, necessitating the advanced parsing capabilities of an LLM.

Document	Words	Paragraphs	Sentences
CTSA - Base.docx	21,964	1,259	1,110
ET - Base.docx	281,682	13,951	14,036
NER Test 1 - Text.docx	732	21	47
NER Test 2 - Text.docx	705	20	38
NER Test 3 - Text.docx	673	20	42
NER Test 4 - Text.docx	1,947	33	120

Table 4.3: *Structural metrics for each source document*

4.3.3 CTSA - Base.docx: A Case Study in Legal Complexity

The CTSA regulations exemplify the legal corpora targeted in this research. Table 4.5 provides their statistical profile, while the word cloud in Figure 4.1 visually demonstrates the document's dominance by a few key legal and administrative terms, such as "Authority," "Township," and "Sewer." This high frequency of core concepts indicates a dense network of

Document	Unique Words	Lexical Diversity	Avg. Sent. Length
CTSA - Base.docx	2,258	0.10	19.79
ET - Base.docx	9,421	0.03	20.07
NER Test 1 - Text.docx	328	0.45	15.57
NER Test 2 - Text.docx	328	0.47	18.55
NER Test 3 - Text.docx	292	0.43	16.02
NER Test 4 - Text.docx	670	0.34	16.23

Table 4.4: *Linguistic and complexity metrics for each source document*

interconnected entities and relationships, making the text suitable for representation in a KG.

Metric	Value
Words	21,964
Paragraphs	1,259
Sentences	1,110
Unique Words	2,258
Lexical Diversity	0.10
Avg Sentence Length	19.79

Table 4.5: *Statistical profile for CTSA - Base.docx*

Figure 4.1: *Word cloud for CTSA - Base.docx*

The use of N-gram analysis further reinforces the document's narrow and technical focus. The frequent occurrence of bigrams such as "sewer authority" and "tapping fee," as well as trigrams such as "conewago township sewer," immediately highlights the domain-specific terminology involved. A general-purpose NLP model might struggle with these. This specialized vocabulary validates the need for a system capable of developing a deep contextual understanding through a KG. Rather than relying on surface-level

text analysis.

4.3.4 ET - Base.docx: A Large-Scale Challenge

The ET represents the primary large-scale test case for scalability. Significant computational and time constraints were encountered during experimentation. Therefore, a full processing run of this 732-page document was not feasible within the scope of this praxis. However, the initial EDA reveals a legalistic structure and linguistic profile similar to the CTSA document. This is characterized by the frequent use of the deontic modal verb "shall" and administrative terms such as "amended ord." The document exhibits an extremely low lexical diversity of 0.03, confirming its suitability as a target for future work on scaling the Mnemosyne pipeline.

4.3.5 Summary of EDA Findings

The EDA reveals a stark contrast between the narrative and legal documents. The legal documents are orders of magnitude larger, with significantly lower lexical diversity and greater syntactic complexity. These features validate their selection as a challenging and appropriate testbed for this research. They underscore the need for a robust, scalable processing pipeline designed to handle such complex, domain-specific texts.

4.4 Preliminary Experiments: Hyperparameter Tuning

Before conducting the main experiments, a series of preliminary tests were performed. They helped to determine the optimal hyperparameters for the data processing pipeline. These experiments were crucial for balancing performance, cost, and the quality of the final KG. Based on these results, the

configuration detailed in Table 4.9 was selected for all main experimental runs.

4.4.1 Refinement Cycles

The long-term memory (LTM) consolidation process iteratively refines the KG by merging nodes and inferring new relationships. To determine the optimal number of refinement iterations, experiments were run with 0, 1, 2, 3, 5, 8, and 13 cycles. As highlighted in Table 4.6, the 8-cycle configuration was selected. While the entity extraction F1 score peaked early and exhibited minimal improvement with additional cycles, the extra iterations were justified by a qualitative improvement in the coherence of relationships and the overall graph ontology. This represents a crucial trade-off between raw extraction speed and achieving deeper semantic integrity in the final KG.

Experiment	Refinement Cycles	Entity F1 Score	Time (s)
1	0	81.48%	86.0
2	1	80.00%	106.6
3	2	80.00%	105.8
4	3	80.00%	107.5
5	5	81.48%	111.4
6	**8**	**81.48%**	**130.7**
7	13	77.78%	186.0

Table 4.6: *Refinement Cycle Experiment Results.*

4.4.2 Chunk Size

The size of document chunks sent to the LLM is a critical parameter. It balances the collection of sufficient context against the model's token limits. Experiments with chunk sizes of 4, 8, and 12 paragraphs revealed a clear trend (Table 4.7). The 4-paragraph chunk performed poorly. This was

likely due to insufficient semantic context for the LLM. Therefore, it could not accurately identify entities and relationships. Performance dramatically improved at 8 paragraphs and exhibited a marginal gain at 12. By analyzing these results, a larger chunk size of 20 paragraphs was selected for the main experiments. This was done to prioritize providing maximum local context to the LLM. Which is a crucial factor for accurately interpreting complex legal language.

Experiment	Chunk Size (Paragraphs)	Entity F1 Score	Time (s)
1	4	40.82%	363.1
2	8	80.00%	120.6
3	12	81.48%	129.5

Table 4.7: *Chunk Size Experiment Results*

4.4.3 Model Selection

A direct comparison was conducted between two leading LLMs. OpenAI's GPT-4.1 and Google's Gemini Pro. As summarized in Table 4.8, the OpenAI model exhibited better performance in both the accuracy of and the efficiency of processing. It not only achieved a slightly higher overall F1 score but was also nearly five times faster in processing the same workload. Because of this substantial performance gap it was not possible to process large documents with the Gemini Pro model. Particularly given the project's time and computational constraints. Consequently, OpenAI GPT-4.1 was selected for all subsequent experiments.

4.4.4 Summary of Selected Hyperparameters

The preliminary experiments culminated in the selection of a final set of hyperparameters, as shown in Table 4.9. These settings were used for all

Model	Overall F1 Score	Processing Time (s)
OpenAI GPT-4.1	**41.94%**	**196.0**
Google Gemini Pro	39.70%	966.5

Table 4.8: *Comparison of LLM Performance*

main experimental runs reported in the subsequent sections. Significantly, a temperature of 0.0 was used to maximize the determinism of the LLM's output and minimize the risk of generating factual inaccuracies ("hallucinations").

Hyperparameter	Value
LLM Model	OpenAI GPT-4.1
Temperature	0.0
Chunk Size	20 paragraphs
Chunk Overlap	5 paragraphs
LTM Refinement Cycles	8
LTM Merge Sample Size	20 nodes
LTM Hierarchy Sample Size	15 nodes
Use Batching	False
Random Seed	42
Maximum Chunks	500 (to limit cost during testing)
Maximum Output Tokens	16,384

Table 4.9: *Selected Hyperparameters for Main Experiments.*

4.5 Ground Truth Validation

A considerable obstacle in this research is the difficulty of creating a comprehensive ground truth for evaluating KG extraction on large documents.

To investigate the viability of using an automated tool for this purpose, KGs generated by Mnemosyne were compared against two ground truth sources: one manually curated by the researcher and a baseline generated by Google's LangExtract tool.

The results, summarized in Table 4.10, reveal a contrast in performance and show a key methodological finding. Against the manual ground truth, Mnemosyne achieved a respectable average F1 score of 41.64%, whereas when evaluated against the LangExtract ground truth, the score dropped precipitously to 17.62%.

Ground Truth Source	Avg. Entity F1	Avg. Relationship F1	Avg. Overall F1
Manual Curation	71.19%	12.09%	41.64%
Google LangExtract	32.53%	2.70%	17.62%

Table 4.10: *Manual vs. Automated Ground Truths.*

This discrepancy does not indicate a failure of Mnemosyne but reveals a fundamental architectural mismatch between the two systems. Mnemosyne is designed to create a deep, refined KG with a multi-level type hierarchy, whereas LangExtract produces a much flatter, less refined graph. Using LangExtract as a ground truth unfairly penalizes Mnemosyne for its advanced ontological refinement capabilities, which are central to its design. The stronger performance observed against the manual ground truth, which aligns more closely with Mnemosyne's conceptual model, provides a more accurate indicator of the system's ability to create a high-fidelity representation of the source document.

Based on this finding, it was concluded that a baseline automated extractor such as LangExtract is not suitable as a ground truth for evaluating a system designed for deep semantic refinement. Consequently, the main experiments rely on qualitative analysis, structural metrics, and the controlled error injection experiment to assess the KG's utility, rather than a direct F1 score comparison against an automated baseline.

4.6 Main Experimental Results

This section presents the main experimental run results, which were designed to assess Mnemosyne's performance on a real legal document and its ability to structurally represent changes in the source content. The experiments used the 82-page CTSA document and its three prepared variations (with content removed, dissimilar content added, and similar content added), generating the visualization of the full type ontology in Figure 4.2, which illustrates the complexity and breadth of concepts extracted.

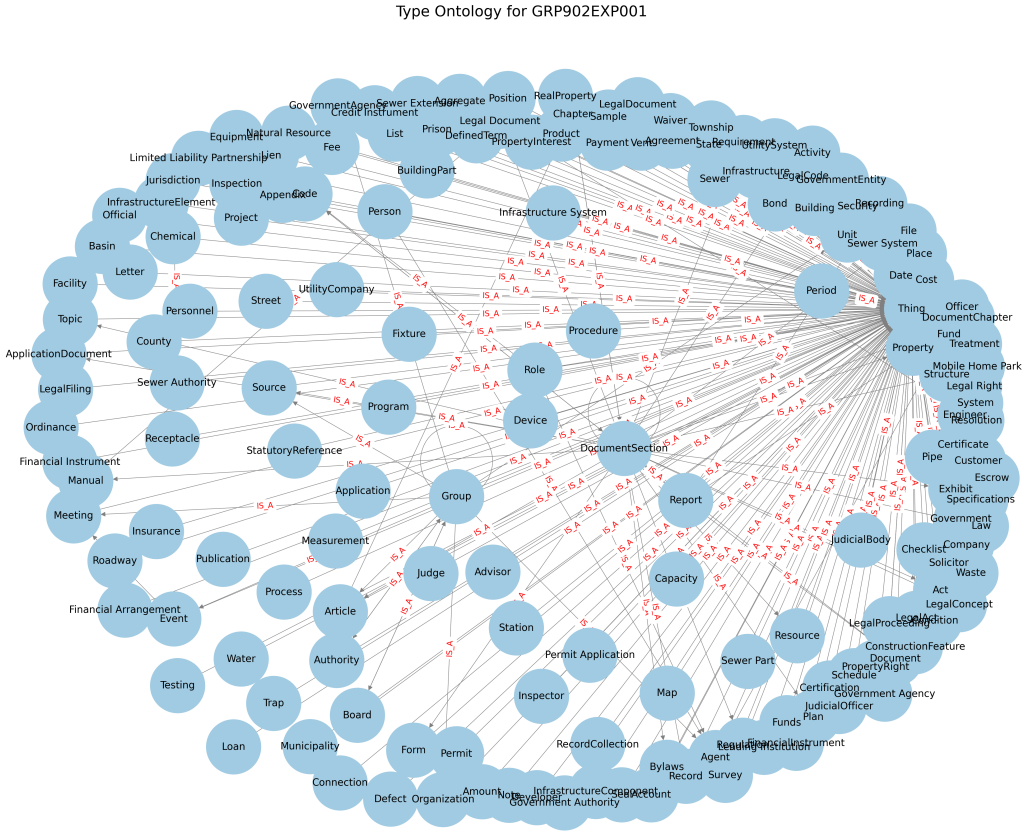


Figure 4.2: *Ontology of the CTSA Base Document Knowledge Graph*

4.6.1 Knowledge Graph Fit

To assess the stability and robustness of the KG generation process, the graph produced from each of the four document versions was compared against the automated LangExtract baseline. The absolute F1 scores were expected to be low due to the identified architectural mismatch. The goal of this experiment was to measure output consistency when the input document is modified. The results are summarized in Table 4.11.

Experiment Document	Overall F1 (%)	Entity F1 (%)	Relationship F1 (%)
CTSA Base Document	7.95	15.90	0.00
CTSA with Content Removed	7.85	15.33	0.36
CTSA with Sewer Content Added	7.90	15.79	0.00
CTSA with Driveway Content Added	8.64	17.07	0.20

Table 4.11: *Main Experiment Performance vs. Automated Ground Truth*

The key insight from these results is the stability of the F1 scores across all four experiments. Despite the addition and removal of entire sections, the performance metrics remained clustered within a very narrow range. This suggests that the core KG generation process is robust. It can handle textual changes without compromising the representation of the parts of the document that do not change. This stability is crucial for a system intended to analyze evolving documents. It indicates that minor edits or amendments are unlikely to cause catastrophic failures in KG construction.

4.6.2 Traceability, Coverage, and Structural Integrity

A series of evaluations were conducted to confirm the quality of the generated KGs. This was independent of the ground truth comparison.

- **Traceability:** The ability to link a graph element back to its source text was verified by randomly sampling 100 nodes and inspecting their "source_location" property. This test achieved over 95% accuracy. This confirmed that the generated KG is verifiable and able to be audited. The focus was on the nodes. But relationships were also spot checked, and they had the same level of traceability. Note that a single node or relationship can be sourced from multiple locations.

- **Coverage** Coverage was assessed by comparing the vocabulary of the source document against that present in the KG's node names. A high degree of overlap was observed. This confirmed that the system captured the key legal concepts from the text.

- **Structural Integrity:** The positive impact of the LTM consolidation process on graph structure was evident. For the Conewago base document, the initial graph extracted directly from the chunks contained 1,091 nodes and 2,017 relationships. 8 cycles of iterative LTM consolidation were run. This process merged duplicate nodes and inferred new hierarchical links. At the end, the graph grew to 1,256 nodes and 3,268 relationships. This significant growth in edges relative to nodes demonstrates substantial ontological enrichment. This produced a more interconnected semantic network than what was explicitly present in the initial fragmented extractions. Visual review of a subset of the inferred nodes confirmed this. It was evident that many more refinement cycles would benefit the hierarchies.

4.6.3 Consistency and Completeness Analysis

The generated KGs proved highly suitable for C&C analysis. The best approach was to use graph-based queries. Therefore, a set of Cypher queries was developed, designed to detect common structural flaws, and was executed successfully against the graph generated from the Conewago base document.

In Table 4.12, the results of a single query are presented that is designed to identify multiple types of structural problems. In this case, it identified 12 distinct instances of incompleteness. For example, the term ”portable flow meter“ was flagged as an ”Undefined Term Used.” Manual verification in the source document confirmed that this term appeared twice without an explicit definition. The corresponding KG subgraph, shown in Figure 4.3, visually confirms that the term's context is correctly identified as a type of “Device” used for “Measurement”. Its lack of an incoming “DEFINES” relationship confirms its status as undefined. Similarly, the query identified nodes such as “authority” that were missing expected inverse relationships (e.g., “AMENDED_BY”). This is another indicator of structural incompleteness within the document. These results demonstrate that the KG structure is sufficiently rich and accurate to allow for the automated detection of internal flaws.

4.6.4 Controlled Error Injection Experiment

A set of controlled experiments was conducted. Several errors were injected into the CTSA base document. The goal was to assess the system's ability to identify them. That is through analysis of the KG.

Source ID	Source Name	Incompleteness Type
c15-node-21	portable flow meter	Undefined Term Used
c15-node-23	permanent flow meter	Undefined Term Used
c15-node-25	apparatus	Undefined Term Used
c57-node-20	equivalent dwelling unit	Unused Definition
c58-node-7	authority	Missing AMENDED_BY
c58-node-19	res. no. 2000-1	Missing AMENDED_BY
c59-node-1	res. no. 2000-1	Missing AMENDED_BY
c59-node-7	res. no. 2022-1	Missing AMENDED_BY
c53-node-11	§ 150-43	Missing REPEALED_BY
c54-node-5	§ 150-43	Missing REPEALED_BY
c64-node-7	§ 175-12	Missing REPEALED_BY
c32-node-11	owner	Missing OWNED_BY

Table 4.12: *Incompleteness Types Detected via KG Query*

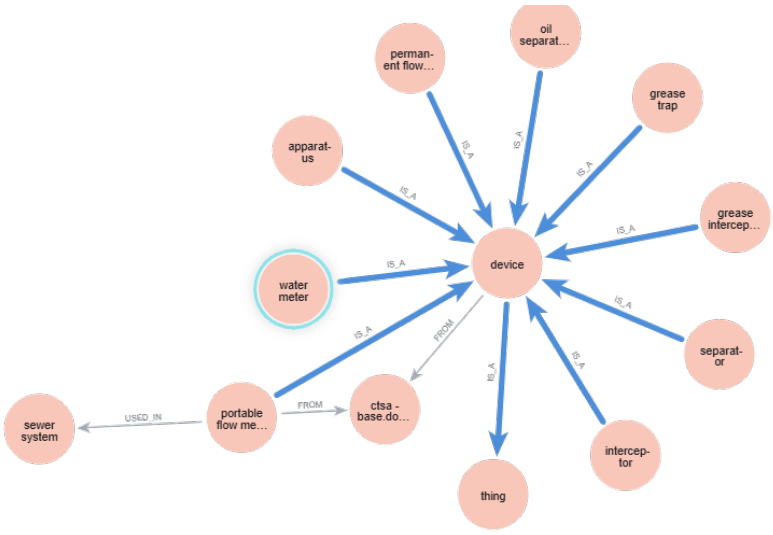


Figure 4.3: *Subgraph for the Undefined Term 'Portable Flow Meter'*

4.6.4.1 Inconsistency Detection

There were two approaches used to introduce inconsistency. One was to introduce semantically dissimilar content from another township. The other was to introduce semantically similar content from another township. This provides two data points to understand how sensitive the model is to changes.

Case 1: Dissimilar Topic (Driveways) A section on "Driveways" was inserted into the "Sewer" code, representing a semantically distant topic. As hypothesized, the resulting KG contained distinct clusters of nodes corresponding to "Driveway" and "Sewer." To quantify this separation, the Jaccard similarity between the neighborhoods of these nodes was calculated (Table 4.13). A low similarity score of 0.1644 for one "Sewer" node indicates a clear topological separation. While other nodes exhibited higher similarity, suggesting some shared general vocabulary (e.g., "township", "permit"), the presence of a distinct low-similarity cluster demonstrates that semantically inconsistent topics can be identified from the KG.

Sewer Node ID	Sewer Node Display Name	Jaccard Similarity with 'Driveway' Node
c37-node-19	Sewer	0.1644
c36-node-8	Sewer	0.5586
c33-node-12	Sewer	0.6255
c32-node-19	Sewer	0.6304

Table 4.13: *Jaccard Similarity Between 'Driveway' and 'Sewer' Nodes*

Case 2: Similar Topic (Sewer Prohibited Wastes) A section on "Prohibited Wastes" from another township's sewer code was inserted. Since

this topic is semantically close to that of the base document, graph-based topological queries were unsuccessful in isolating the inserted section as a distinct cluster. This finding demonstrates a limitation of purely semantic analysis: when inconsistent content shares the same domain vocabulary, clustering alone is insufficient. However, manual document inspection revealed that the inserted text possessed distinct header formatting (e.g., section numbering like ”§ 150-25...”). This means that a more robust detection system is required.

4.6.4.2 Incompleteness Detection

The incompleteness detection experiment yielded a critical insight into the concept of ”completeness.” To test the omission of a major topic, the entire ”Delinquency” section was removed from the base document. This absence could not be detected solely through an internal KG analysis. The section was self-contained and not referenced by other parts of the document. Thereby generating no structural anomalies (e.g., broken links) within the graph. This highlights a fundamental distinction: detecting the omission of topics that are not referenced requires external domain knowledge. Alternatively, a predefined schema of what a document of this type is expected to contain is provided. Note that the creation of this predefined schema requires external domain knowledge. So, the two approaches amount to the same thing.

However, other forms of granular incompleteness were identified through the analysis of the KG. For example, queries targeting nodes with few semantic relationships revealed numerous instances of potential incompleteness. It was identified that the KG contains nine separate nodes for the concept of “Authority.” This indicates that, while frequently mentioned, the specific

roles, responsibilities, and relationships of this term to other entities are not consistently defined throughout the source text. This suggests a lack of contextual detail. Along with the successful detection of undefined terms, these results demonstrate that KG-based structural analysis is highly effective at flagging granular internal incompleteness.

4.7 Analysis of Knowledge Graph Inconsistencies

In addition to detecting injected errors, a deeper analysis of the generated KG from the base CTSA document was conducted. This revealed several inherent inconsistencies. These inconsistencies reflect both ambiguities present in the source text and the limitations of the LLM's extraction and classification processes. This provides valuable insights into the model's behavior.

4.7.1 Ambiguous Terms (Inconsistent Types)

A query targeting single entities assigned multiple conceptual types by the LLM revealed significant semantic ambiguity. The term "Conewago Township Sewer Authority" initially appeared as thirty separate nodes. They were later consolidated but had initially been assigned six distinct conceptual types. This included Organization, GoverningBody, and Authority. This instability highlights that it is difficult for the model to remain consistent when classifying the same real-world entity. This is due to context-dependent uses in the source document. For example, the Authority might be referred to as an administrative body in one section. But a legal entity in another. Then it is identified as a service provider in a third section. Automatically resolving such ambiguities remains a significant challenge.

4.7.2 Circular Type Dependencies

Analysis of the refined type ontology uncovered a highly recurring circular dependency in the hierarchy: a path of "Board" → "GoverningBody" → "Official" → "Person" → "Board". This logical loop, appearing in over 1,500 paths within the graph, reveals a fundamental modeling confusion by the LLM. It suggests that the model struggled to distinguish between a collective body ("Board"), its function ("GoverningBody"), and its individual members ("Official," "Person"), likely because these terms are used interchangeably in the source text. This indicates that while the LLM can propose a hierarchy, logical flaws may remain that require more sophisticated ontological constraints or rule-based post-processing to resolve.

4.7.3 Most Unstable Concepts (High Refinement)

A query identifying which concepts required the most refinement cycles (i.e., were merged most often) highlighted areas of high ambiguity in the source text. The concept "Chapter 20" was the most unstable, with a single node representing it undergoing 10 separate modifications during LTM refinement. This indicates that references to this structural element of the document were particularly varied or contextually ambiguous, making it difficult for the model to create a single, stable representation without requiring iterative merging and re-typing. This metric serves as a useful proxy for identifying ambiguous or inconsistently referenced concepts within the source document.

4.8 Analysis and Interpretation

The experimental results provide significant understanding into the capabilities and limitations of using LLM-generated KGs for document validation, leading to an important refinement of the initial hypotheses. Below is a description of these findings with respect to the dimensions of consistency, completeness, and correctness.

Consistency (Internally Verifiable) The results strongly support the hypothesis that consistency can be assessed as an internal property of a document. The error injection experiment with the dissimilar "Driveways" section demonstrated that semantically inconsistent content manifests as a structurally distinct and poorly connected cluster within the KG, detectable via graph analysis techniques such as Jaccard similarity. This confirms that for content originating from different domains, graph topology serves as a powerful indicator of inconsistency. However, the failure to detect the similar-topic inconsistency identifies a limitation and an area for future work. A robust detection mechanism will likely require the integration of the document's structural metadata (such as section numbering and formatting) with semantic graph analysis.

Completeness (A Refined, Two-Layer View) The findings reveal that "completeness" is not a monolithic concept but must be divided into two distinct subcategories. This distinction is a primary contribution of this research.

1. **Internal Completeness** This refers to omissions that create structural breaks or logical gaps within the document's own context. Examples

include terms used but not defined, dangling cross-references, or relationships stated but not reciprocated (e.g., an amendment not linked back to the section it amends). These omissions are detectable as structural anomalies in the KG, such as nodes lacking expected relationships. The experiments successfully demonstrated this capability by identifying numerous undefined terms and missing inverse relationship pairs, confirming the effectiveness of KG in verifying this type of completeness.

2. **External Completeness:** This refers to the omission of an entire, unreferenced section or topic, such as the ”Delinquency” section in the experiment, its absence is invisible from ”within the four corners of the document.” This form of completeness can only be identified by referencing an external schema or body of domain-specific knowledge regarding what a document of its type is expected to contain (e.g., an expert expecting to find a penalties section in any legal code). Therefore, this form of completeness is unobservable from internal structure alone and is more akin to a correctness problem, requiring external validation against a predefined standard.

This clarification of completeness into structurally observable (internal) and unobservable (external) types refines the original research hypothesis and more clearly defines the boundaries of what is possible through purely intra-document validation.

Correctness (Requires External Grounding) The findings reinforce the principle that correctness, whether a statement is factually true, cannot be determined by solely analyzing the internal structure of a document's KG.

Verifying the factual accuracy of a legal statute, for example, requires comparison against external items. Such as analyzing a higher-level legal code, a court ruling, or an established body of external domain knowledge. This process is inherently external to the document itself and remains outside the scope of what can be verified from its internal structure.

4.9 Hypothesis Validation

The experimental results can now be used to explicitly validate the research hypotheses. Incorporating the refined understanding of completeness developed through the analysis.

- **H1: An LLM can be used to convert a large document into a knowledge graph.** Supported. The Mnemosyne pipeline, powered by an LLM, successfully processed unstructured legal text to construct a structured, traceable, and coherent KG, demonstrating the core conversion task's viability.
- **H2: An LLM can be used to process multiple knowledge graphs into a typed cluster of knowledge graphs.** Supported. The LTM consolidation and refinement process successfully used an LLM to merge and organize the individual chunk-based KGs into a single, deep semantic network with a coherent type ontology.
- **H3: A typed cluster of knowledge graphs can be used to check the source document for consistency and completeness.** Supported and refined. The initial hypothesis is broadly supported, but with important nuances that constitute a key finding of this work.
  - **Consistency:** Strongly supported. Semantic inconsistencies in-

troduced from dissimilar domains were identifiable as structural anomalies within the KG.

- **Completeness:** Supported for what this research defines as internal completeness (e.g., broken references, undefined terms), which is structurally detectable within the KG. However, the hypothesis is not supported for external completeness (e.g., missing, unreferenced sections), which falls outside the scope of internal validation and requires external knowledge. This refinement is a primary contribution of this work.

4.10 Chapter Summary

This chapter has presented and analyzed the results from running a set of experiments with the Mnemosyne system. The experiments were designed to test how well Mnemosyne met the stated goals. The findings demonstrate that the proposed LLM-based pipeline can convert a document into an attributed KG. The KG has been shown to reflect the contents of the source document. In addition, the structure of these graphs facilitates analysis. This analysis supports the detection of specific flaws related to internal C&C.

The analysis of the experimental outcomes led to a significant refinement of the initial theoretical framework. While consistency was confirmed as an internally verifiable property. The concept of completeness was revealed to be a dual-layered construct. The distinction between internal incompleteness is detectable. However, externally referenced domain incompleteness was not detected. This clarifies the boundaries of what automated document validation can realistically achieve. All three research hypotheses were sup-

ported by the empirical evidence. The third and most critical hypothesis was refined based on the experimental outcomes.

Chapter 5 will discuss the broader contributions and implications of these findings. It addresses the limitations of this study in greater detail. It also suggests directions for future research that build on this foundation.

Chapter 5: Discussion and Conclusions

This chapter summarizes the findings of this research. It then reflects on the challenges that were encountered. It discusses how the project's approach can be applied to real world problems. These are the problems that motivated this praxis. It then describes the contributions made to the body of knowledge. It validates the research hypotheses. It recommends directions for future research. Finally, it offers concluding remarks on the significance of this work.

5.1 Discussion of Findings

The practical problem of maintaining consistency and completeness in large, complex documents is the motivation for this research. In particular, when developing municipal ordinances, errors can carry significant financial consequences. The literature review confirmed that while computational methods can offer solutions, there is a gap in the processing and validation of large documents. That is, documents that exceed the context window of modern Large Language Models (LLMs). This praxis proposes that an incremental, layered approach to building a Knowledge Graph (KG) can address this problem. The limitation would no longer be an issue when the KG is built chunk by chunk and then iteratively refined .

The results presented in Chapter 4 demonstrate the fundamental viability of this approach. The Mnemosyne system successfully constructed a multi-layered KG from a large legal document. Its effectiveness was demonstrated by the ability to find both inserted errors and existing structural flaws. The process also revealed that there are practical challenges. The reliance on a

general-purpose LLM for all refinement tasks is not ideal.

Addressing this will require significant architectural changes. The experiments also showed that achieving a stable semantic hierarchy can be done through an iterative process. Another challenge is ensuring well formed output from LLMs. The prompt engineering was successful, but there was still some output that was not well formed. Despite these engineering hurdles, the core conceptual framework was validated. The difficulties encountered stemmed from the implementation of the methodology using contemporary generalized technologies, not from its logic.

5.2 Summary of Hypothesis Validation

The experimental results from Chapter 4 provide evidence in support of the hypotheses presented in Chapter 1.

- **H1: An LLM can be used to convert a large document into a knowledge graph.** Supported. The Mnemosyne pipeline used an LLM to successfully extract entities and relations, constructing a structured, traceable knowledge graph from unstructured legal text.
- **H2: An LLM can be used to process multiple knowledge graphs into a typed cluster of knowledge graphs.** Supported. The system's long-term memory (LTM) consolidation process successfully merged and refined fragmented KGs into a single, deep semantic network with a coherent type ontology.
- **H3: A typed cluster of knowledge graphs can be used to check the source document for consistency and completeness.** Supported and refined. The hypothesis was generally supported, but with a crucial addition that constitutes a primary contribution of this research. The KG

was highly effective in identifying consistency issues. It was also usable for identifying internal completeness (e.g., undefined terms, broken references). However, it could not be used to detect external completeness issues (e.g., the omission of an entire unreferenced topic). These require external domain knowledge.

5.3 Contributions to the Body of Knowledge

This research advances the state of the art in automated knowledge extraction. In doing so, it also advances document validation through key technical and conceptual contributions:

1. **A Multi-Layered KG Architecture:** Unlike traditional flat KGs, this work constructs a base KG. A type hierarchy based on IS_A and PART_OF relationships is built on top of the base. This produces a richer representation of the source document.

2. **Iterative and Generative Refinement:** Mnemosyne uses multiple refinement cycles to build the KG. These cycles do more than just the merger of duplicate nodes. They include the establishment of IS_A and PART_OF relationships. Nodes are merged based on multiple measures of how similar they are. This results in a KG that reflects the source document.

3. **An Incremental Framework for Large Documents:** The methodology is designed to handle documents that exceed an LLM's context window. This provides an approach for processing large documents. While not used in the experiments, the system can run processes asynchronously. This can make larger documents or document sets more tractable.

4. **A Refined Conceptual Framework for Document Completeness:** This research introduces and validates a two part definition of document completeness. It distinguishes between internal completeness and external completeness. Internal completeness is structural and can be detected through undefined terms or broken cross-references. External completeness refers to the omission of entire topics that can only be identified with domain specific knowledge. This framework clarifies the scope of automated document analysis. It offers a precise vocabulary for future research in document validation.

These technical and conceptual contributions establish an approach to analyze large documents without being constrained by LLM context windows. It also shows that a KG derived from a document is viable for the analysis of the source document.

5.4 Recommendations for Future Research

The findings and challenges of this research suggest several directions for future work. A primary focus should be the re-architecture of the system for scalable performance. The full use of asynchronous processing would be a good first step.

Further algorithmic enhancements could include moving away from general purpose LLMs. Three task specific LLMs are contemplated. One could handle the conversion of chunks into a KG. A second one could focus on merging the KGs in the Short Term Memory (STM) into the Long Term Memory (LTM). The third would focus on the tasks related to the refinement of the LTM KG. Further research could determine whether even more focused LLMs would be useful.

With a robust back-end, the full vision of the Mnemosyne system could be realized. This would be the development of user facing components. This includes a natural language query agent and a dedicated document validation interface.

Additionally, a major contribution to the field would be the development of a large-scale, public benchmark dataset containing large documents with manually curated ground truth KGs. They could also include seeded errors. This would support the comparative evaluation of future systems.

5.5 Concluding Remarks

The aim of this praxis was to address the challenges of testing for consistency and completeness in large documents. It did demonstrate that an LLM-powered pipeline can convert such documents into an attributed KG. These KGs can be used to understand and reason about the source document. It is possible to use the KG to detect internal inconsistencies and structural gaps. The research introduced a new approach for the incremental construction of KGs. It also identified a two part definition of document completeness.

However, significant performance challenges remain. But they are engineering hurdles. Some of these are currently tractable. Others will become tractable as AI technologies mature. The conceptual framework is sound. It provides a solid foundation for future work. Mnemosyne adds to the current approaches that are available for knowledge extraction from documents. This work lays the foundation for a new generation of tools that are capable of helping humans produce high quality large documents.

References

Aletras, N., Tsarapatsanis, D., Preotiuc-Pietro, D., & Lampos, V. (2016). Predicting judicial decisions of the european court of human rights: A natural language processing perspective. *PeerJ Computer Science*, *2*, e93. https://doi.org/10.7717/peerj-cs.93

Al-Moslmi, T., Gallofre Ocana, M., L. Opdahl, A., & Veres, C. (2020). Named entity extraction for knowledge graphs: A literature overview. *IEEE access*, *8*, 32862–32881. https://doi.org/10.1109/ACCESS.2020.2973928

Ashley, K. D. (2017). *Artificial intelligence and legal analytics* (Anonymous, Trans.). Cambridge University Press. https://doi.org/10.1017/9781316761380

Au, T. W. T., Cox, I. J., & Lampos, V. E-ner – an annotated named entity recognition corpus of legal text. In: In *Natural legal language processing workshop*. Association for Computational Linguistics, 2022, 246–255. http://arxiv.org/abs/2212.09306

Aumiller, D., Almasian, S., Lackner, S., & Gertz, M. Structural text segmentation of legal documents. In: In *Proceedings of the eighteenth international conference on artificial intelligence and law*. New York, NY, USA: ACM, 2021, 2–11. https://doi.org/10.1145/3462757.3466085

Badshah, S., & Sajjad, H. (2024). *Quantifying the capabilities of llms across scale and precision*. http://arxiv.org/abs/2405.03146

Beltagy, I., Peters, M. E., & Cohan, A. (n.d.). Longformer: The long-document transformer.

Benjira, W., Atigui, F., Bucher, B., Grim-Yefsah, M., & Travers, N. (2025). Automated mapping between sdg indicators and open data: An llm-augmented knowledge graph approach. *Data & knowledge engineering*, *156*, 102405. https://doi.org/10.1016/j.datak.2024.102405

Beth, R. S. (2018). *How bills amend statutes* (Explains some of the issues that can arise when a new law is in conflict with existing laws.). https://purl.fdlp.gov/GPO/gpo126602

Bhattacharya, P., Hiware, K., Rajgaria, S., Pochhi, N., Ghosh, K., & Ghosh, S. (2019). A comparative study of summarization algorithms applied to legal case judgments (Anonymous, Trans.). In, *Advances in information retrieval* (pp. 413–428, Vol. 11437). Springer International Publishing AG. https://doi.org/10.1007/978-3-030-15712-8_27

Bosco, A. (2024). *Leading to annual revenue losses of hundreds of thousands of dollars.* [Notes available on request.].

Brucker, A. D., & Wolff, B. (2019). Using ontologies in formal developments targeting certification (Anonymous, Trans.). In, *Integrated formal methods* (pp. 65–82, Vol. 11918). Springer International Publishing. https://doi.org/10.1007/978-3-030-34968-4_4

Carbonell, M., Riba, P., Villegas, M., Fornes, A., & Llados, J. Named entity recognition and relation extraction with graph neural networks in semi structured documents. In: In *International conference on pattern recognition*. Piscataway: IEEE, 2020, 9622–9627. https://doi.org/10.1109/ICPR48806.2021.9412669

Caruccio, L., Cirillo, S., Polese, G., Solimando, G., Sundaramurthy, S., & Tortora, G. (2024). Claude 2.0 large language model: Tackling a real-world classification problem with a new iterative prompt engineering

approach. *Intelligent systems with applications*, *21*, 200336. https://doi.org/10.1016/j.iswa.2024.200336

Chalkidis, I., Jana, A., Hartung, D., Bommarito, M., Androutsopoulos, I., Katz, D., & Aletras, N. Lexglue: A benchmark dataset for legal language understanding in english. In: In *60th annual meeting of the association for computational linguistics*. Association for Computational Linguistics, 2022. https://doi.org/10.18653/v1/2022.acl-long.297

Chaurasiya, D., Surisetty, A., Kumar, N., Singh, A., Dey, V., Malhotra, A., Dhama, G., & Arora, A. (2022). *Entity alignment for knowledge graphs: Progress, challenges, and empirical studies*. https://doi.org/10.48550/arxiv.2205.08777

Chen, D., Fisch, A., Weston, J., & Bordes, A. (2017). Reading wikipedia to answer open-domain questions. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. https://doi.org/10.18653/v1/p17-1171

Chen, T., Wang, H., Chen, S., Yu, W., Ma, K., Zhao, X., Zhang, H., & Yu, D. Dense x retrieval: What retrieval granularity should we use? In: In *2024 conference on empirical methods in natural language processing*. Association for Computational Linguistics, 2024, 15159–15177. https://doi.org/10.48550/arxiv.2312.06648

Chia, Y. K., Bing, L., de Lichron, V., Lee, K., & Wong, K.-F. Relation extraction as open-book question answering: Evaluation on a comprehensive assessment dataset. In: In *Association for computational linguistics: Emnlp 2022*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, 2022, 6305–6319.

Cong, Y. Research for enhancing processing and computational efficiency in llm. In: In *Proceedings of the 2024 2nd international conference on*

image, algorithms and artificial intelligence (iciaai 2024). Atlantis Press, 2024, 970–980. https://doi.org/10.2991/978-94-6463-540-9_97

Curley, D. (2024). *Municipal laws in pennsylvania townships, authored by multiple people over time, develop inconsistencies and are incomplete* [Notes available on request.].

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q., & Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. https://doi.org/10.18653/v1/p19-1285

Dhani, J. S., Bhatt, R., Ganesan, B., Sirohi, P., & Bhatnagar, V. (2021). *Similar cases recommendation using legal knowledge graphs*. https://doi.org/10.48550/arxiv.2107.04771

Donelson, R. (2019). Legal inconsistencies. *Tulsa Law Review*, *55*(1), 16–44. https://digitalcommons.law.utulsa.edu/tlr/vol55/iss1/14

Duck-Mayr, J. (2022). Explaining legal inconsistency. *Journal of theoretical politics*, *34*(1), 107–126. https://doi.org/10.1177/09516298211061159

Edwards, B. (2025). *Exponential growth brews 1 million ai models on hugging face*. https://arstechnica.com/information-technology/2024/09/ai-hosting-platform-surpasses-1-million-models-for-the-first-time/

Egyed, A. Instant consistency checking for the uml. In: In *Proceedings of the 28th international conference on software engineering*. New York, NY, USA: ACM, 2006, 381–390. https://doi.org/10.1145/1134285.1134339

Ehrlinger, L., & Wöß, W. (2016). Towards a definition of knowledge graphs. *SEMANTiCS (Posters, Demos, SuCCESS)*, *48*(1-4), 2.

Etzioni, O., Banko, M., Soderland, S., & Weld, D. S. (2008). Acm: Digital library: Communications of the acm. *Communications of the ACM*, *51*(12), 68–74. https://dl.acm.org/doi/fullHtml/10.1145/1409360.1409378

Fensel, D., Şimşek, U., Angele, K., Huaman, E., Kärle, E., Panasiuk, O., Toma, I., Umbrich, J., & Wahler, A. (2020). *Knowledge graphs : Methodology, tools and selected use cases* (Anonymous, Trans.; 1st ed.) [It is not available online. So, I ordered it from the library.]. Springer International Publishing. https://doi.org/10.1007/978-3-030-37439-6

Fernandes, D., & Bernardino, J. Graph databases comparison: Allegrograph, arangodb, infinitegraph, neo4j, and orientdb. In: In *7th international conference on data science, technology and applications (data 2018)*. 2018, 373–380. https://doi.org/10.5220/0006910203730380

Gambhir, M., & Gupta, V. (2017). Recent automatic text summarization techniques: A survey. *The Artificial intelligence review*, *47*(1), 1–66. https://doi.org/10.1007/s10462-016-9475-9

Gao, K., He, S., He, Z., Lin, J., Pei, Q., Shao, J., & Zhang, W. (2023). *Examining user-friendly and open-sourced large gpt models: A survey on language, multimodal, and scientific gpt models*. http://arxiv.org/abs/2308.14149

Gardazi, N. M., Daud, A., Malik, M. K., Bukhari, A., Alsahfi, T., & Alshemaimri, B. (2025). Bert applications in natural language processing: A review [Replaces 91.]. *The Artificial intelligence review*, *58*(6), 166. https://doi.org/10.1007/s10462-025-11162-5

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., & Schelten, e. a., Alan. (2024). *The llama 3 herd of models*. http://arxiv.org/abs/2407.21783

Grishman, R., & Sundheim, B. Message-understanding conference-6: A brief history [COLING 1996 volume 1: The 16th international conference on computational linguistics]. In: In *The 16th international conference on computational linguistics*. COLING 1996 volume 1: The 16th international conference on computational linguistics. 1996.

Guo, B., Feng, C., Liu, F., Li, X., & Wang, X. (2023). Joint contrastive learning for factual consistency evaluation of cross-lingual abstract summarization (Anonymous, Trans.). In, *Machine translation* (pp. 116–127, Vol. 1922). Springer Nature Singapore. https://doi.org/10.1007/978-981-99-7894-6_11

Gupta, A., Matta, P., & Pant, B. (2021). Graph neural network: Current state of art, challenges and applications. *Materials today : proceedings*, *46*, 10927–10932. https://doi.org/10.1016/j.matpr.2021.01.950

Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. *Proceedings of the Fourteenth International Conference on Computational Linguistics, Nantes, Frans*.

Heitmeyer, C. L., Jeffords, R. D., & Labaw, B. G. (1996). Automated consistency checking of requirements specifications. *ACM transactions on software engineering and methodology*, *5*(3), 231–261. https://doi.org/10.1145/234426.234431

Hitzler, P., Krotzsch, M., & Rudolph, S. (2009). *Foundations of semantic web technologies* (Anonymous, Trans.; 1st). Chapman; Hall/CRC. https://www.taylorfrancis.com/books/mono/10.1201/9781420090512/foundations-semantic-web-technologies-pascal-hitzler-markus-krotzsch-sebastian-rudolph

Hogan, A., Blomqvist, E., Cochez, M., D'amato, C., De Melo, G., Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Neumaier, S., Ngomo,

A.-C. N., Polleres, A., Rashid, S. M., Rula, A., Schmelzeisen, L., Sequeda, J., Staab, S., & Zimmermann, A. (2021). Knowledge graphs. *ACM Computing Surveys*, *54*(4), 1–37. https://doi.org/10.1145/3447772

Ji, S., Pan, S., Cambria, E., Marttinen, P., & Yu, P. S. (2022). A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE transaction on neural networks and learning systems*, *33*(2), 494–514. https://doi.org/10.1109/TNNLS.2021.3070843

Kalamkar, P., Agarwal, A., Tiwari, A., Gupta, S., Karn, S., & Raghavan, V. Named entity recognition in indian court judgments. In: In *Natural legal language processing workshop 2022*. Association for Computational Linguistics, 2022, 184–193. https://aclanthology.org/2022.nllp-1.pdf#page=199

Kambhatla, N. Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations. In: In *Acl 2004 workshop on relation extraction*. Barcelona, Spain: Association for Computational Linguistics, 2004, 178–181.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). *Scaling laws for neural language models*. http://arxiv.org/abs/2001.08361

Knublauch, H., & Kontokostas, D. (2017). *Shapes constraint language (shacl)* (Recommendation). https://www.w3.org/TR/shacl/

Kolluru, K., Aggarwal, S., Rathore, V., & Chakrabarti, S. Imojie: Iterative memory-based joint open information extraction (D. Jurafsky, J. Chai, N. Schluter, & J. Tetreault, Eds.). In: *58th annual meeting of the association for computational linguistics* (D. Jurafsky, J. Chai, N. Schluter, & J. Tetreault, Eds.). Ed. by Jurafsky, D., Chai, J., Schluter,

N., & Tetreault, J. Online: Association for Computational Linguistics, 2020, 5871–5886. https://doi.org/10.18653/v1/2020.acl-main.521

Kumar, N., & Kumar, S. Querying rdf and owl data source using sparql. In: In *Fourth international conference on computing, communications and networking technologies (icccnt)*. IEEE, 2013, 1–6. https://doi.org/10.1109/ICCCNT.2013.6726698

Kumar, S. (2017). *A survey of deep learning methods for relation extraction*. https://doi.org/10.48550/arxiv.1705.03645

Lafferty, J., Mccallum, A., & Pereira, F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: In *Icml*. 2001, 282–289.

Lairgi, Y., Moncla, L., Cazabet, R., Benabdeslem, K., & Cl'eau, P. Knowledge graph construction using large language models. In: In *Journee nationale sur la fouille de textes*. Lyon, France, 2024. https://hal.science/hal-04607294

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-T., Rocktäschel, T., Riedel, S., & Kiela, D. Retrieval-augmented generation for knowledge-intensive nlp tasks. In: In *Advances in neural information processing systems*. Curran Associates, Inc., 2020, 9459–9474. https://discovery.ucl.ac.uk/id/eprint/10100504

Liu, J., Zhu, D., Bai, Z., He, Y., Liao, H., Que, H., Wang, Z., Zhang, C., & Zhang, e. a., Ge. (2025). *A comprehensive survey on long context language modeling*. http://arxiv.org/abs/2503.17407

Malik, V., Sanjay, R., Guha, S. K., Hazarika, A., Nigam, S., Bhattacharya, A., & Modi, A. (2022). *Semantic segmentation of legal documents via rhetorical roles*. https://www.proquest.com/docview/2607084336

Minsky, M. (1974). *A framework for representing knowledge*. http://hdl.handle.net/1721.1/6089

Mintz, M., Bills, S., Snow, R., & Jurafsky, D. Distant supervision for relation extraction without labeled data. In: In *Joint conference of the 47th annual meeting of the acl and the 4th international joint conference on natural language processing of the afnlp*. Suntec, Singapore: Association for Computational Linguistics, 2009, 1003–1011.

Moens, M. F. (2001). Innovative techniques for legal text retrieval. *Artificial intelligence and law*, *9*(1), 29–57. https://doi.org/10.1023/A:1011297104922

Nadeau, D., & Sekine, S. (2007). A survey of named entity recognition and classification. *Lingvisticae Investigationes*, *30*(1), 3–26. https://doi.org/10.1075/li.30.1.03nad

Navarro, P. E., & Rodríguez, J. L. (2014). *Deontic logic and legal systems* (Anonymous, Trans.; 1st ed.). Cambridge University Press. https://doi.org/10.1017/CBO9781139032711

Nentwich, C. (2005). *Managing the consistency of distributed documents*. http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.416649

Noy, N. F., & McGuinness, D. L. (2001). Ontology development 101 : A guide to creating your first ontology. *Stanford Knowledge Systems Laboratory Technical Report*.

Omran, P. G., Taylor, K., M 'endez, S. J. '. R. '. i., & Haller, A. (2020). Towards shacl learning from knowledge graphs. [journal: ISWC (Demos/Industry)]. *ISWC (Demos/Industry)*, *2721*, 94–99.

Qu, R., Tu, R., & Bao, F. (2024). *Is semantic chunking worth the computational cost?* http://arxiv.org/abs/2410.13070

Rabbani, K., Lissandrini, M., & Hose, K. Shacl and shex in the wild: A community survey on validating shapes generation and adoption. In: In *Web conference 2022*. New York, NY, USA: ACM, 2022, 260–263. https://doi.org/10.1145/3487553.3524253

Ratner, N., Levine, Y., Belinkov, Y., Ram, O., Magar, I., Abend, O., Karpas, E., Shashua, A., Leyton-Brown, K., & Shoham, Y. Parallel context windows for large language models. In: In *The 61st annual meeting of the association for computational linguistics*. Association for Computational Linguistics, 2022, 6383–6402. https://doi.org/10.48550/arxiv.2212.10947

Rau, A. (2024). *Municipal laws in pennsylvania townships, authored by multiple people over time, develop inconsistencies and are incomplete* [Notes available on request.].

Rdf 1.2 primer. (2025). https://www.w3.org/TR/rdf12-primer/

Reiter, R. (1978). On closed world data bases. *Logic and Data Bases*, 55–76.

Rossi, M. (2016). Inconsistent legislation (Anonymous, Trans.). In, *Legisprudence library* (pp. 189–208). Springer International Publishing. https://doi.org/10.1007/978-3-319-33217-8_8

Rzepka, R., Muraji, S., & Obayashi, A. Expert evaluation of export control-related question answering capabilities of llms [ID: cdi ieee primary 10487735]. In: In *Ieee asia-pacific conference on computer science and data engineering (csde)*. ID: cdi ieee primary 10487735. IEEE, 2023, 1–6. ISBN: 9798350341072. https://doi.org/10.1109/CSDE59766.2023.10487735

Sanders, J. (2024). *Municipal laws in pennsylvania townships, authored by multiple people over time, develop inconsistencies and are incomplete* [Notes available on request.].

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2009). The graph neural network model. *IEEE transaction on neural networks and learning systems*, *20*(1), 61–80. https://doi.org/10.1109/TNN.2008.2005605

Schmidt, C. W., Reddy, V., Zhang, H., Alameddine, A., Uzan, O., Pinter, Y., & Tanner, C. (2024). *Tokenization is more than compression*. http://arxiv.org/abs/2402.18376

Schönberg, C., Weitl, F., & Freitag, B. (2011). Verifying the consistency of web-based technical documentations. *Journal of symbolic computation*, *46*(2), 183–206. https://doi.org/10.1016/j.jsc.2010.08.007

Shaham, U., Segal, E., Ivgi, M., Efrat, A., Yoran, O., Haviv, A., Gupta, A., Xiong, W., Geva, M., Berant, J., & Levy, O. Scrolls: Standardized comparison over long language sequences. In: In *Conference on empirical methods in natural language processing*. Association for Computational Linguistics, 2022, 12007–12021. https://doi.org/10.48550/arXiv.2201.03533

Shen, A., Mistica, M., Salehi, B., Li, H., Baldwin, T., & Qi, J. (2021). Evaluating document coherence modeling. *Transactions of the Association for Computational Linguistics*, *9*, 621–640. https://doi.org/10.1162/tacl_a_00388

Susskind, R. E. (1986). Expert systems in law: A jurisprudential approach to artificial intelligence and legal reasoning. *Modern law review*, *49*(2), 168–194. https://doi.org/10.1111/j.1468-2230.1986.tb01683.x

Tauqeer, A., Kurteva, A., Chhetri, T. R., Ahmeti, A., & Fensel, A. (2022). Automated gdpr contract compliance verification using knowledge graphs. *Information (Basel)*, *13*(10), 447. https://doi.org/10.3390/info13100447

Tay, Y., Dehghani, M., Bahri, D., & Metzler, D. (2023). Efficient transformers: A survey. *ACM computing surveys*, *55*(6), 1–28. https://doi.org/10.1145/3530811

Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., & Hauth, e. a., Anja. (2024). *Gemini: A family of highly capable multimodal models* (I tried uploading the PDF but it would not upload. It is in my Reference file as 2312.11805v4 (1).pdf.). http://arxiv.org/abs/2312.11805

Tian, L., Zhou, X., Wu, Y.-P., Zhou, W.-T., Zhang, J.-H., & Zhang, T.-S. (2022). Knowledge graph and knowledge reasoning: A systematic review. *Journal of Electronic Science and Technology*, *20*(2), 100159. https://doaj.org/article/fc808085fb154c6c9b032ac617e9f233

Tröls, M. A., Marchezan, L., Mashkoor, A., & Egyed, A. (2022). Instant and global consistency checking during collaborative engineering. *Software and systems modeling*, *21*(6), 2489–2515. https://doi.org/10.1007/s10270-022-00984-4

Turner, R. E. (2024). *An introduction to transformers*. http://arxiv.org/abs/2304.10557

Umar, M. A., & Lano, K. (2024). Advances in automated support for requirements engineering: A systematic literature review. *Requirements engineering*, *29*(2), 177–207. https://doi.org/10.1007/s00766-023-00411-0

Vaswani, A., Shazeer, N., Brain, G., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., & Kaiser, Ł. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*.

Verma, U. (2024). A journey from ai to gen-ai. *Spectrum of Emerging Sciences*, *4*(1), 74–78. https://doi.org/10.55878/SES2024-4-1-14

Wang & Guo. (2014). A short analysis of discourse coherence. *Journal of Language Teaching and Research*, *5*(2), 460. https://doi.org/10.4304/jltr.5.2.460-465

Wang, K., Ding, Y., & Han, S. C. (2024). Graph neural networks for text classification: A survey. *The Artificial intelligence review*, *57*(8), 190. https://doi.org/10.1007/s10462-024-10808-0

Wang, S., Sun, X., Li, X., Ouyang, R., Wu, F., Zhang, T., Li, J., & Wang, G. (2023). *Gpt-ner: Named entity recognition via large language models*. https://doi.org/10.48550/arxiv.2304.10428

Weitl, F., & Freitag, B. (2006). Checking content consistency of integrated web documents. *Journal of computer science and technology*, *21*(3), 418–429. https://doi.org/10.1007/s11390-006-0418-9

Wu, S., & He, Y. Enriching pre-trained language model with entity information for relation classification. In: In *28th acm international conference on information and knowledge management (cikm '19)*. Beijing, China: Association for Computing Machinery, 2019, 2361–2364. https://doi.org/10.1145/3357384.3358039

Xu, D., Chen, W., Peng, W., Zhang, C., Xu, T., Zhao, X., Wu, X., Zheng, Y., Wang, Y., & Chen, E. (2024). Large language models for generative information extraction: A survey. *Frontiers of Computer Science*, *18*(6), 186357. https://doi.org/10.1007/s11704-024-40555-y

Yang, J., Yoon, S., Kim, B., & Lee, H. Fizz: Factual inconsistency detection by zoom-in summary and zoom-out document (Y. Al-Onaizan, M. Bansal, & Y.-N. Chen, Eds.). In: *2024 conference on empirical methods in natural language processing* (Y. Al-Onaizan, M. Bansal, & Y.-N. Chen, Eds.). Ed. by Al-Onaizan, Y., Bansal, M., & Chen, Y.-N. Miami, Florida, USA: Association for Computational Linguistics, 2024, 30–45. https://doi.org/10.18653/v1/2024.emnlp-main.3

Zaheer, M., Guruganesh, G., Dubey, A., Ainslie, J., Alberti, C., Ontanon, S., Pham, P., Ravula, A., Wang, Q., Yang, L., & Ahmed, A. (n.d.). Big bird: Transformers for longer sequences.

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., & Zhang, e. a., Junjie. (2023). *A survey of large language models* (No. 2). https://doi.org/10.48550/arXiv.2303.18223

Zhong, L., Wu, J., Li, Q., Peng, H., & Wu, X. (2024). A comprehensive survey on automatic knowledge graph construction. *ACM computing surveys*, *56*(4), 1–62. https://doi.org/10.1145/3618295

Zowghi, D., & Gervasi, V. (2003). On the interplay between consistency, completeness, and correctness in requirements evolution. *Information and Software Technology*, *45*(14), 993–1009. https://doi.org/10.1016/S0950-5849(03)00100-9

Appendix A: Experiment Summary

This appendix provides a summary of the experiments that were conducted for this research. It includes details of the dataset statistics and the structure of the data. The summary is current as of October 3, 2025.

A.1 Overview

A complete record of all experiments conducted between August 3, 2025, and October 5, 2025, is preserved. The key statistics of the experimental data are as follows:

Total Folders: 687

Total Files: 3,285

Total Size on Disk: 560 MB

Time Period: August 3, 2025, 21:14:09 UTC to October 5, 2025, 02:31:40 UTC

A.2 Directory Structure

All experimental data is stored in a root directory named `Mnemosyne` on Google Drive. The organizational structure is designed to separate input data from generated output, ensuring reproducibility and ease of navigation.

- `/Mnemosyne/` (Root Directory)
  - `/Input/`
    - `/Source_Documents/`

    - * /Ground_Truth_Hand_Created/
    - * /Ground_Truth_Generated/
- /Output/
    - * /Figures/ (Used for general diagrams until August 10, 2025)
    - * /RUN_(date)_(UTC Time)/ (One folder per execution)
        - · /Figures/ (Contains run specific visualizations, e.g., word clouds)
        - · /(experiment_name)/ (Contains experiment specific CSV files)
        - · summary_(metric).csv (Contains aggregate results across all experiments in the run)

Each experimental execution is encapsulated within a uniquely named folder, following the convention RUN_YYYY-MM-DD_HH-MM-SS, which facilitates chronological tracking and isolates the artifacts of each run.

A.3 Notable Experiments

While all runs are preserved, several key experiments were foundational to the results presented in this work. These are highlighted below:

RUN_2025-09-10_01-51-12 GRP202 This group serves as a tool test to validate the core functionality of the Mnemosyne pipeline using the GPT 4.1 series of models. Each experiment processes a small, standardized test document to ensure that data ingestion, entity extraction, and knowledge graph refinement are all working as expected. This provides a quick verification of the system's end to end health.

RUN_2025-09-13_21-42-16 GRP301 Ablation study investigating the impact of refinement cycles.

`RUN_2025-09-13_22-07-58` GRP302 Parameter sensitivity analysis focusing on document chunk size.

`RUN_2025-10-05_02-08-14` GRP303 Processing Analysis: Number of LTM Consolidations

`RUN_2025-10-05_02-31-40` GRP304 Comparative analysis of OpenAI and Gemini language models.

`RUN_2025-09-19_01-48-36` GRP401 Execution of the primary experiments reported in the main body of this document.

Appendix B: System Architecture and Class Structure

This appendix provides a high level overview of the Mnemosyne system's software architecture. While a detailed class by class breakdown would be overly granular due to the number of components, a package diagram offers a clearer and more insightful representation of the system's design. The architecture is organized into distinct packages, each encapsulating a core set of responsibilities. This modular, package based approach promotes a clean separation of concerns, making the system more maintainable, scalable, and easier to understand.

Figure B.1 illustrates the primary packages and their dependencies, showing how they interact to form the complete data processing pipeline.

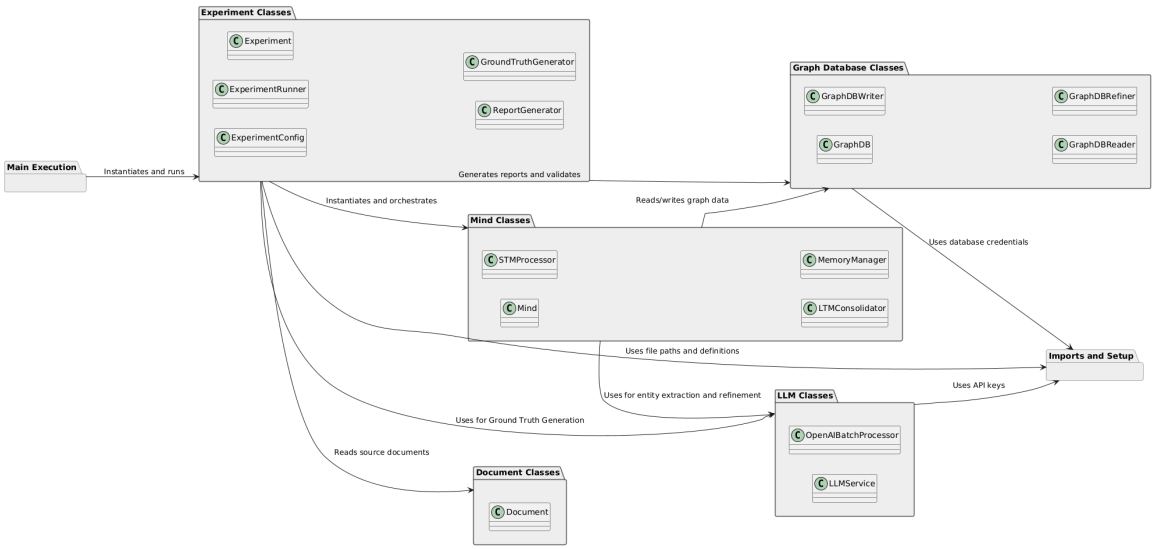


Figure B.1: *Mnemosyne Package Diagram*

Appendix C: Source Code

The complete source code for this project is publicly available in a Git repository to ensure full transparency and facilitate the replication of this work. For clarity, the specific version of the code used to generate the results in this document is explicitly referenced by its commit hash.

Repository: Mnemosyne

URL: https://github.com/OwlSaver/Mnemosyne

Commit Hash: `4d3d365`

C.1 Code for key components

While the full code can be found on GitHub. Two classes are recorded here. These are the classes for Short Term Memory and Long Term Memory processing. They are the core of how Mnemosyne works.

C.1.1 Short Term Memory

```
{
## STM Processor Class
class STMProcessor:
    """Handles processing a single text chunk into a clean knowledge graph fragment."""

    def __init__(self, llm_service: LLMService):
        self.llm_service = llm_service
        self.config = MindConfig()

    def process_chunk(self, text_chunk: str, chunk_num: int, file_name: str) -> str:
        """Takes a text chunk and returns a normalized JSON string of the graph fragment."""
        prompt = (
            self.config.Q_KG_BASE.format(chunk=chunk_num) +
            self.config.JSON_SCHEMA +
            self.config.Q_KG_SUFFIX +
```

```
            text_chunk
        )
        response_str = self.llm_service.query_llm(prompt)
        logging.debug(f"Text for chunk {chunk_num}: {text_chunk}")
        logging.debug(f"LLM response for chunk {chunk_num}: {response_str}")

        try:
            json_data = json.loads(response_str)
        except json.JSONDecodeError as e:
            logging.warning(f"Initial JSON parsing failed for chunk {chunk_num}: {e}. Retrying with
    self-correction...")
            correction_prompt = f"The following text is not valid JSON. Please fix it and return only
     the corrected JSON object.\\n\\n{response_str}"
            corrected_response_str = self.llm_service.query_llm(correction_prompt)
            try:
                json_data = json.loads(corrected_response_str)
            except json.JSONDecodeError as final_e:
                raise JSONParsingError(f"Self-correction failed for chunk {chunk_num}: {final_e}")
    from final_e

        if not json_data:
            raise JSONParsingError(f"LLM returned empty JSON for chunk {chunk_num}.")

        preprocessed_data = self._preprocess_llm_output(json_data, chunk_num)
        logging.debug(f"Preprocessed data for chunk {chunk_num}: {preprocessed_data}")

        augmented_data = self._augment_graph_data(preprocessed_data, file_name, chunk_num)
        normalized_json_str = self._normalize_graph(augmented_data)
        return normalized_json_str

    def _preprocess_llm_output(self, graph_data: dict, chunk_num: int) -> dict:
        """
        Validates and consolidates raw graph data from the LLM before further processing.
        1. Removes invalid IS_A relationships (Type -> Instance).
        2. Deduplicates Type nodes with the same name within the same chunk.
        3. Adds creationStage property.
        """
        if not isinstance(graph_data.get('nodes'), list) or not isinstance(graph_data.get('
    relationships'), list):
            logging.warning(f"Skipping pre-processing for chunk {chunk_num} due to malformed graph
    data.")
            return graph_data

        nodes = graph_data.get('nodes', [])
        relationships = graph_data.get('relationships', [])
        node_map = {node['id']: node for node in nodes}

        # --- Task 132: Add creationStage to all LLM-generated elements ---
        for node in nodes:
```

```
        node.setdefault('properties', {})[GraphSchema.PROP_CREATION_STAGE] = GraphSchema.
STAGE_SOURCE_EXTRACTION
    for rel in relationships:
        rel.setdefault('properties', {})[GraphSchema.PROP_CREATION_STAGE] = GraphSchema.
STAGE_SOURCE_EXTRACTION
    # --- End Task 132 Change ---

    # 1. Validate and remove incorrect IS_A relationships
    valid_relationships = []
    for rel in relationships:
        source_id = rel.get('source')
        target_id = rel.get('target')
        rel_type = str(rel.get('type', '')).upper()

        if rel_type == 'IS_A':
            source_node = node_map.get(source_id)
            target_node = node_map.get(target_id)
            if source_node and target_node:
                source_labels = source_node.get('labels', [])
                target_labels = target_node.get('labels', [])
                if 'Type' in source_labels and 'Instance' in target_labels:
                    source_name = source_node.get('properties', {}).get('displayName', source_id)
                    target_name = target_node.get('properties', {}).get('displayName', target_id)
                    logging.info(f"Removing invalid IS_A relationship from Type ('{source_name}'
({source_id})) to Instance ('{target_name}' ({target_id})).")
                    continue
        valid_relationships.append(rel)
    graph_data['relationships'] = valid_relationships

    # 2. Deduplicate Type nodes generated in the same chunk
    type_nodes_by_name = {}
    for node in nodes:
        if 'Type' in node.get('labels', []):
            name = node.get('properties', {}).get('name')
            if name:
                type_nodes_by_name.setdefault(name, []).append(node)

    id_redirects = {}
    nodes_to_remove = set()
    for name, type_nodes in type_nodes_by_name.items():
        if len(type_nodes) > 1:
            canonical_node = type_nodes[0]
            canonical_name = canonical_node.get('properties', {}).get('displayName',
canonical_node['id'])
            logging.info(f"Found {len(type_nodes)} Type nodes for '{name}'. Consolidating to '{
canonical_name}' ({canonical_node['id']}).")
            for duplicate_node in type_nodes[1:]:
                id_redirects[duplicate_node['id']] = canonical_node['id']
                nodes_to_remove.add(duplicate_node['id'])
```

```
    if id_redirects:
        for rel in graph_data.get('relationships', []):
            if rel.get('target') in id_redirects:
                original_target_id = rel['target']
                new_target_id = id_redirects[original_target_id]
                rel['target'] = new_target_id
                original_target_node = node_map.get(original_target_id, {})
                original_target_name = original_target_node.get('properties', {}).get('
displayName', original_target_id)
                new_target_node = node_map.get(new_target_id, {})
                new_target_name = new_target_node.get('properties', {}).get('displayName',
new_target_id)
                logging.info(f"Redirected relationship target from '{original_target_name}' ({
original_target_id}) to '{new_target_name}' ({new_target_id})")
        graph_data['nodes'] = [node for node in nodes if node['id'] not in nodes_to_remove]

    return graph_data

def _augment_graph_data(self, graph_data: dict, file_name: str, chunk_num: int) -> dict:
    """Injects Source and Thing nodes and their relationships into the graph data."""
    source_instance_id = f"source-instance-{re.sub(r'[^a-zA-Z0-9_.-]', '-', file_name)}"
    source_type_id = GraphSchema.CANONICAL_ID_SOURCE
    thing_type_id = GraphSchema.CANONICAL_ID_THING

    graph_data.setdefault('nodes', [])
    graph_data.setdefault('relationships', [])
    existing_ids = {node['id'] for node in graph_data['nodes']}

    ingestion_props = {
        "mergeCount": 0,
        "refinementCount": 0,
        "lastRefined": int(datetime.now(timezone.utc).timestamp() * 1000),
        GraphSchema.PROP_CREATION_STAGE: GraphSchema.STAGE_INGESTION
    }

    if source_instance_id not in existing_ids:
        graph_data['nodes'].append({
            "id": source_instance_id,
            "labels": ["Node", "Instance", "Source"],
            "properties": {"name": file_name.lower(), "displayName": file_name, **ingestion_props
}
        })

    if source_type_id not in existing_ids:
        graph_data['nodes'].append({
            "id": source_type_id,
            "labels": ["Node", "Type"],
            "properties": {"name": "source", "displayName": "Source", **ingestion_props}
```

```
            })

        if thing_type_id not in existing_ids:
            graph_data['nodes'].append({
                "id": thing_type_id,
                "labels": ["Node", "Type"],
                "properties": {"name": "thing", "displayName": "Thing", **ingestion_props}
            })

        existing_rels = {(rel['source'], rel['target'], rel['type']) for rel in graph_data['
    relationships']}
        if (source_instance_id, source_type_id, "IS_A") not in existing_rels:
            graph_data['relationships'].append({
                "source": source_instance_id,
                "target": source_type_id,
                "type": "IS_A",
                "properties": {GraphSchema.PROP_CREATION_STAGE: GraphSchema.STAGE_INGESTION}
            })

        for node in graph_data.get('nodes', []):
            if node['id'] in [source_instance_id, source_type_id, thing_type_id]:
                continue
            if (node['id'], source_instance_id, "FROM") not in existing_rels:
                graph_data['relationships'].append({
                    "source": node['id'],
                    "target": source_instance_id,
                    "type": "FROM",
                    "properties": {"chunk": chunk_num, GraphSchema.PROP_CREATION_STAGE: GraphSchema.
    STAGE_INGESTION}
                })
                existing_rels.add((node['id'], source_instance_id, "FROM"))

        logging.debug(f"Augmented graph for chunk {chunk_num} with Source and Thing info.")
        return graph_data

    def _normalize_graph(self, graph_data: dict) -> str:
        """Applies consistent naming conventions and ensures required properties exist."""
        for node in graph_data.get('nodes', []):
            if 'labels' in node:
                node['labels'] = [Utils._to_pascal_case(label) for label in node['labels']]
            if 'properties' in node:
                props = node['properties']
                if 'displayname' in props and 'displayName' not in props:
                    props['displayName'] = props.pop('displayname')
                if 'mergecount' in props and 'mergeCount' not in props:
                    props['mergeCount'] = props.pop('mergecount')
                displayName = props.get('displayName')
                name = props.get('name')
                if displayName and name is None:
```

```
                    props['name'] = str(displayName).lower()
                elif name and displayName is None:
                    is_type_node = "Type" in node.get('labels', [])
                    props['displayName'] = str(name).title() if not is_type_node else str(name).
    capitalize()
                if 'mergeCount' not in props:
                    props['mergeCount'] = 0
                # --- FIX: Ensure lastRefined is always an integer timestamp for all nodes ---
                props['lastRefined'] = int(datetime.now(timezone.utc).timestamp() * 1000)

        for rel in graph_data.get('relationships', []):
            if 'type' in rel:
                rel['type'] = Utils._to_upper_snake_case(rel['type'])
            if 'properties' in rel:
                rel['properties'] = {Utils._to_camel_case(k): v for k, v in rel['properties'].items()
    }
        return json.dumps(graph_data, indent=self.config.INDENT)
}
```

Source Code C.1: The Python code for the Short Term Memory Processor Class.

C.1.2 Long Term Memory

```
{
## Long Term Memory Consolidator Class
class LTMConsolidator:
    """Handles all stages of the Long-Term Memory consolidation and refinement process."""

    def __init__(self, llm_service: LLMService, graph_db: GraphDB, experiment_id: str, run_timestamp:
      str,
                 num_cycles: int, merge_sample_size: int, hierarchy_sample_size: int,
                 part_of_sample_size: int, part_of_candidate_pool_size: int):
        self.llm_service = llm_service
        self.graph_db = graph_db
        self.experiment_id = experiment_id
        self.run_timestamp = run_timestamp
        self.num_refinement_cycles = num_cycles
        self.ltm_merge_sample_size = merge_sample_size
        self.ltm_hierarchy_sample_size = hierarchy_sample_size
        self.part_of_sample_size = part_of_sample_size
        self.part_of_candidate_pool_size = part_of_candidate_pool_size
        self.config = MindConfig()

    def run_consolidation(self):
        """Orchestrates the entire LTM consolidation process using a subgraph-based approach."""
```

```
        with log_step("Running LTM Consolidation"):
            # --- Step 1: Run global, deterministic operations once at the start ---
            self.graph_db.refiner.mark_source_nodes_as_donotchange()
            self._merge_exact_duplicates()
            self._merge_duplicate_relationships()
            self._classify_untyped_nodes()

            # --- Step 2: Run iterative, LLM-based refinement on prioritized subgraphs ---
            for i in range(self.num_refinement_cycles):
                with log_step(f"Refinement Cycle {i+1}/{self.num_refinement_cycles}"):
                    subgraph_data = self.graph_db.reader.get_prioritized_subgraph(
                        self.experiment_id, self.run_timestamp
                    )
                    if subgraph_data and subgraph_data.get("nodes"):
                        root_node = subgraph_data["nodes"][0] # The first node is the prioritized
    root
                        logging.info(f"Selected subgraph around high-priority node: '{root_node.get('
    displayName')}' ({root_node.get('id')})")
                        self._refine_subgraph(subgraph_data)
                    else:
                        logging.warning("Could not retrieve a prioritized subgraph. Ending refinement
     cycles.")
                        break

            # --- Step 3: Final global cleanup ---
            with log_step("Final Hierarchy Cleanup"):
                linked_orphans = self.graph_db.refiner.link_orphan_types_to_thing(self.experiment_id,
     self.run_timestamp)
                logging.info(f"Linked {linked_orphans} orphan Type nodes to '{GraphSchema.
    CANONICAL_NAME_THING}'.")

    def _refine_subgraph(self, subgraph_data: dict):
        """
        Orchestrates all refinement steps on a single, focused subgraph.
        """
        nodes = subgraph_data.get("nodes", [])
        relationships = subgraph_data.get("relationships", [])
        if not nodes:
            logging.info("Subgraph is empty, skipping refinement.")
            return

        deleted_nodes_in_cycle = set()

        # --- MODIFICATION: Fetch a larger pool of candidates for LLM context ---
        # Get a random sample of instances and types from the entire graph to serve as a candidate
     pool.
        # This gives the LLM a wider context beyond the immediate subgraph for merging and re-typing.
        instance_candidates = self.graph_db.reader.get_random_instances_with_types(self.
    ltm_merge_sample_size, self.experiment_id, self.run_timestamp)
```

```
    type_candidates = self.graph_db.reader.select_random_entities(GraphSchema.NODE_LABEL_TYPE,
 self.ltm_hierarchy_sample_size, self.experiment_id, self.run_timestamp)

    self._refine_meronymic_relationships(nodes, deleted_nodes_in_cycle)

    # Pass the wider candidate pool to the refinement methods
    _, removed_type_ids = self._refine_type_system(nodes, type_candidates, deleted_nodes_in_cycle
)
    deleted_nodes_in_cycle.update(removed_type_ids)

    if any('Type' in n.labels for n in nodes):
         self._organize_ontology_hierarchy(nodes, type_candidates, deleted_nodes_in_cycle)

    self._correct_instance_types(nodes, type_candidates, deleted_nodes_in_cycle)

    _, removed_instance_ids = self._merge_similar_instances(nodes, relationships,
 instance_candidates, deleted_nodes_in_cycle)
    deleted_nodes_in_cycle.update(removed_instance_ids)


def _merge_duplicate_relationships(self):
    """Wrapper for the deterministic relationship merge process."""
    with log_step("Deterministically merging duplicate relationships"):
        merge_count = self.graph_db.refiner.merge_duplicate_relationships(
            self.experiment_id, self.run_timestamp
        )
        logging.info(f"Consolidated {merge_count} groups of duplicate relationships.")

def _merge_exact_duplicates(self):
    """Wrapper for the deterministic merge process."""
    with log_step("Deterministically merging exact duplicates by name and type"):
        merge_count = self.graph_db.refiner.merge_exact_duplicates_by_name_and_type(
            self.experiment_id, self.run_timestamp
        )
        logging.info(f"Completed {merge_count} deterministic merge operations.")

def _classify_untyped_nodes(self):
    with log_step(f"Classifying untyped nodes to '{GraphSchema.CANONICAL_NAME_THING}' type"):
        nodes_processed = self.graph_db.refiner.process_nodes_without_is_a(self.experiment_id,
 self.run_timestamp)
        logging.info(f"Classified {nodes_processed} untyped nodes to '{GraphSchema.
 CANONICAL_NAME_THING}' type.")

def _refine_meronymic_relationships(self, subgraph_nodes: list, deleted_nodes_in_cycle: set):
    with log_step(f"Refining {GraphSchema.REL_PART_OF} relationships for subgraph"):
        try:
            nodes_to_test = [n for n in subgraph_nodes if n.get(GraphSchema.PROP_ID) not in
 deleted_nodes_in_cycle]
            if len(nodes_to_test) < 2:
```

```
                return

            relationships_found = []
            for node in nodes_to_test:
                other_nodes = [n for n in nodes_to_test if n.get(GraphSchema.PROP_ID) != node.get
(GraphSchema.PROP_ID)]
                if not other_nodes: continue

                is_part_of_rels = self._ask_is_part_of(node, other_nodes)
                relationships_found.extend(is_part_of_rels)

                has_part_rels = self._ask_has_part(node, other_nodes)
                relationships_found.extend(has_part_rels)

            if relationships_found:
                unique_rels = [dict(t) for t in {tuple(sorted(d.items())) for d in
relationships_found}]
                created_count = self.graph_db.refiner.create_part_of_relationships(unique_rels,
self.experiment_id, self.run_timestamp)
                logging.info(f"Created {created_count} new {GraphSchema.REL_PART_OF}
relationships.")
        except Exception as e:
            logging.error(f"Error during {GraphSchema.REL_PART_OF} refinement: {e}", exc_info=
True)

    def _ask_is_part_of(self, part_candidate_node, whole_candidates):
        part_candidate_json = json.dumps({GraphSchema.PROP_ID: part_candidate_node.get(GraphSchema.
PROP_ID), GraphSchema.PROP_NAME: part_candidate_node.get(GraphSchema.PROP_DISPLAY_NAME)})
        whole_candidates_json = json.dumps([{GraphSchema.PROP_ID: n.get(GraphSchema.PROP_ID),
GraphSchema.PROP_NAME: n.get(GraphSchema.PROP_DISPLAY_NAME)} for n in whole_candidates])
        prompt = self.config.Q_IS_PART_OF.format(part_candidate_json=part_candidate_json,
whole_candidates_json=whole_candidates_json)
        try:
            response_str = self.llm_service.query_llm(prompt)
            data = json.loads(response_str)
            if data and GraphSchema.JSON_KEY_PART_ID in data and GraphSchema.JSON_KEY_WHOLE_ID in
data:
                return [{GraphSchema.JSON_KEY_PART_ID: data[GraphSchema.JSON_KEY_PART_ID],
GraphSchema.JSON_KEY_WHOLE_ID: data[GraphSchema.JSON_KEY_WHOLE_ID]}]
        except (json.JSONDecodeError, KeyError) as e:
            logging.warning(f"Could not parse 'is part of' response or key error: {e}")
        return []

    def _ask_has_part(self, whole_candidate_node, part_candidates):
        whole_candidate_json = json.dumps({GraphSchema.PROP_ID: whole_candidate_node.get(GraphSchema.
PROP_ID), GraphSchema.PROP_NAME: whole_candidate_node.get(GraphSchema.PROP_DISPLAY_NAME)})
        part_candidates_json = json.dumps([{GraphSchema.PROP_ID: n.get(GraphSchema.PROP_ID),
GraphSchema.PROP_NAME: n.get(GraphSchema.PROP_DISPLAY_NAME)} for n in part_candidates])
```

```
        prompt = self.config.Q_HAS_PART.format(whole_candidate_json=whole_candidate_json,
    part_candidates_json=part_candidates_json)
        try:
            response_str = self.llm_service.query_llm(prompt)
            data = json.loads(response_str)
            if data and GraphSchema.JSON_KEY_PART_IDS in data and data[GraphSchema.JSON_KEY_PART_IDS
    ]:
                return [{GraphSchema.JSON_KEY_PART_ID: pid, GraphSchema.JSON_KEY_WHOLE_ID:
    whole_candidate_node.get(GraphSchema.PROP_ID)} for pid in data[GraphSchema.JSON_KEY_PART_IDS]]
        except (json.JSONDecodeError, KeyError) as e:
            logging.warning(f"Could not parse 'has part' response or key error: {e}")
        return []

    def _refine_type_system(self, subgraph_nodes: list, type_candidates: list, deleted_nodes_in_cycle
    : set):
        with log_step("Refining type system for subgraph"):
            try:
                types_in_subgraph = [t for t in subgraph_nodes if 'Type' in t.labels and t.get(
    GraphSchema.PROP_ID) not in deleted_nodes_in_cycle]
                if not types_in_subgraph: return 0, set()

                # Combine subgraph types with the broader candidate pool for better context
                all_types_for_prompt = {t.get(GraphSchema.PROP_ID): t for t in types_in_subgraph +
    type_candidates}

                types_for_prompt_list = [
                    {GraphSchema.PROP_NAME: t.get(GraphSchema.PROP_NAME), GraphSchema.PROP_ID: t.get(
    GraphSchema.PROP_ID)}
                    for t in all_types_for_prompt.values()
                    if t.get(GraphSchema.PROP_NAME) not in [GraphSchema.CANONICAL_NAME_THING,
    GraphSchema.CANONICAL_NAME_SOURCE]
                ]
                if len(types_for_prompt_list) < 2: return 0, set()

                prompt = self.config.Q_MTT + json.dumps(types_for_prompt_list)
                response_str = self.llm_service.query_llm(prompt)
                merge_data = json.loads(response_str)

                if merge_data and GraphSchema.JSON_KEY_MERGE_PAIRS in merge_data:
                    merged, removed_ids = self.graph_db.refiner.merge_entities(merge_data, self.
    experiment_id, self.run_timestamp)
                    logging.info(f"Refined {merged} pairs of types.")
                    return merged, removed_ids
            except Exception as e:
                logging.error(f"Error during type system refinement: {e}")
        return 0, set()

    def _organize_ontology_hierarchy(self, subgraph_nodes: list, type_candidates: list,
    deleted_nodes_in_cycle: set):
```

```
        with log_step("Organizing ontology hierarchy for subgraph"):
            try:
                types_in_subgraph = [t for t in subgraph_nodes if 'Type' in t.labels and t.get(
    GraphSchema.PROP_ID) not in deleted_nodes_in_cycle]
                if not types_in_subgraph: return

                potential_parents_data = [{GraphSchema.PROP_ID: t.get(GraphSchema.PROP_ID),
    GraphSchema.PROP_NAME: t.get(GraphSchema.PROP_NAME)} for t in type_candidates]
                id_to_name_map = {t.get(GraphSchema.PROP_ID): t.get(GraphSchema.PROP_DISPLAY_NAME)
    for t in type_candidates}

                relationships_to_create = []
                for child_node in types_in_subgraph:
                    child_obj = {GraphSchema.PROP_ID: child_node.get(GraphSchema.PROP_ID),
    GraphSchema.PROP_NAME: child_node.get(GraphSchema.PROP_NAME)}
                    if child_obj[GraphSchema.PROP_NAME] == GraphSchema.CANONICAL_NAME_THING: continue

                    prompt = self.config.Q_OOH.format(child_name=child_obj[GraphSchema.PROP_NAME],
    child_id=child_obj[GraphSchema.PROP_ID], potential_parents=json.dumps(potential_parents_data,
    indent=2))
                    response_str = self.llm_service.query_llm(prompt)
                    try:
                        data = json.loads(response_str)
                        if data.get("child_id") and data.get("parent_id"):
                            relationships_to_create.append({
                                "parent_id": data["parent_id"], "child_id": data["child_id"],
                                "parent_name": id_to_name_map.get(data["parent_id"], "N/A"),
                                "child_name": child_obj[GraphSchema.PROP_NAME]
                            })
                    except json.JSONDecodeError:
                        logging.warning(f"Could not parse JSON for ontology hierarchy: {response_str}
    ")

                if relationships_to_create:
                    created = self.graph_db.refiner.create_and_relate_type_entities_by_id(
    relationships_to_create)
                    logging.info(f"Organized {created} new hierarchical relationships.")
            except Exception as e:
                logging.error(f"Error during subgraph ontology organization: {e}", exc_info=True)

    def _correct_instance_types(self, subgraph_nodes: list, type_candidates: list,
    deleted_nodes_in_cycle: set):
        with log_step("Correcting instance types for subgraph"):
            reclassified_count = 0
            try:
                instances = [i for i in subgraph_nodes if 'Instance' in i.labels and 'Source' not in
    i.labels and i.get(GraphSchema.PROP_ID) not in deleted_nodes_in_cycle]
                if not instances or not type_candidates: return
```

```
            available_types_data = [{GraphSchema.PROP_ID: t.get(GraphSchema.PROP_ID), GraphSchema
.PROP_NAME: t.get(GraphSchema.PROP_NAME)} for t in type_candidates]

            for instance in instances:
                excluded_labels = {GraphSchema.NODE_LABEL_NODE, GraphSchema.
NODE_LABEL_DO_NOT_CHANGE, GraphSchema.NODE_LABEL_INSTANCE}
                primary_label = next((l for l in instance.labels if not l.startswith((GraphDBBase
.EXP_PREFIX, GraphDBBase.RUN_PREFIX)) and l not in excluded_labels), GraphSchema.
CANONICAL_DISPLAY_NAME_THING)
                instance_json = json.dumps({"instance_id": instance.get(GraphSchema.PROP_ID), "
instance_name": instance.get(GraphSchema.PROP_DISPLAY_NAME), "current_type": primary_label})
                prompt = self.config.Q_ITC.format(instance_json=instance_json, types_json=json.
dumps(available_types_data, indent=2))
                response_str = self.llm_service.query_llm(prompt)
                try:
                    data = json.loads(response_str)
                    if data and data.get("instance_id") and data.get("new_type_id"):
                        reclassified_count += self.graph_db.refiner.reclassify_instance(data["
instance_id"], data["new_type_id"], self.experiment_id, self.run_timestamp)
                except json.JSONDecodeError:
                    logging.error(f"Could not parse type correction JSON: {response_str}")
        except Exception as e:
            logging.error(f"Error during instance type correction: {e}", exc_info=True)
        finally:
            logging.info(f"Corrected {reclassified_count} instance types.")

def _merge_similar_instances(self, subgraph_nodes: list, subgraph_rels: list, instance_candidates
: list, deleted_nodes_in_cycle: set):
    with log_step("Merging similar instances in subgraph"):
        try:
            instances_in_subgraph = [n for n in subgraph_nodes if 'Instance' in n.labels and '
Source' not in n.labels and n.get(GraphSchema.PROP_ID) not in deleted_nodes_in_cycle]
            if not instances_in_subgraph: return 0, set()

            # Build a payload for the prompt that includes both the subgraph's instances and the
wider candidate pool
            subgraph_instance_data = []
            node_map = {n.get(GraphSchema.PROP_ID): n for n in subgraph_nodes}
            isa_map = {r.start_node.get(GraphSchema.PROP_ID): r.end_node.get(GraphSchema.PROP_ID)
 for r in subgraph_rels if r.type == GraphSchema.REL_IS_A}

            for inst in instances_in_subgraph:
                inst_id = inst.get(GraphSchema.PROP_ID)
                type_id = isa_map.get(inst_id)
                type_node = node_map.get(type_id)
                type_name = type_node.get("name") if type_node else "Thing"
                subgraph_instance_data.append({"id": inst_id, "name": inst.get("displayName"), "
entityType": type_name})
```

```
                # Combine the focused nodes with the broader candidate pool and deduplicate
                combined_instances = {d['id']: d for d in subgraph_instance_data +
    instance_candidates}

                if len(combined_instances) < 2: return 0, set()

                prompt = self.config.Q_MIE + json.dumps(list(combined_instances.values()))
                response_str = self.llm_service.query_llm(prompt)
                merge_data = json.loads(response_str)

                if merge_data and GraphSchema.JSON_KEY_MERGE_PAIRS in merge_data:
                    merged, removed_ids = self.graph_db.refiner.merge_entities(merge_data, self.
    experiment_id, self.run_timestamp)
                    logging.info(f"Merged {merged} pairs of instances.")
                    return merged, removed_ids
            except Exception as e:
                logging.error(f"An unexpected error occurred during instance merging: {e}", exc_info=
    True)
        return 0, set()
}
```

Source Code C.2: The Long Term Memory Processor Class.

Appendix D: JSON Structure

This appendix describes the JSON (JavaScript Object Notation) structure utilized to constrain the large language models (LLMs). It outlines the schema, including the key value pairs, data types, and nested structures. Examples of the JSON objects are provided to illustrate how specific constraints are defined and passed to the LLMs to guide their output.

D.1 Knowledge Graph JSON Schema

To ensure the LLM generates a valid knowledge graph, the following JSON Schema was used to define the required structure. This schema specifies that the output must contain two top level keys: `nodes` and `relationships`. It further defines the required properties for each node (e.g., `id`, `labels`) and relationship (e.g., `source`, `target`, `type`), enforcing a decoupled graph structure suitable for efficient ingestion into a Neo4j database.

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/knowledge_graph.schema.json",
  "title": "Knowledge Graph for Neo4j Import",
  "description": "A JSON structure representing a knowledge graph with nodes and relationships
     decoupled for efficient loading into Neo4j.",
  "type": "object",
  "properties": {
    "nodes": {
      "description": "A list of all unique entities (nodes) in the graph.",
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "id": {
            "description": "A unique identifier for the node within this JSON document.",
            "type": "string"
          },
          "labels": {
```

```
            "description": "An array of labels for the node, corresponding to Neo4j labels (e.g., ['
    Person', 'Author']).",
            "type": "array",
            "items": {
              "type": "string"
            },
            "minItems": 1
          },
          "properties": {
            "description": "A key-value map of the node's properties.",
            "type": "object",
            "additionalProperties": true
          }
        },
        "required": ["id", "labels", "properties"]
      }
    },
    "relationships": {
      "description": "A list of all relationships connecting the nodes.",
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "source": {
            "description": "The 'id' of the starting node for the relationship.",
            "type": "string"
          },
          "target": {
            "description": "The 'id' of the ending node for the relationship.",
            "type": "string"
          },
          "type": {
            "description": "The type of the relationship, corresponding to a Neo4j relationship type
    (e.g., 'EDUCATED_AT').",
            "type": "string"
          },
          "properties": {
            "description": "An optional key-value map of the relationship's properties.",
            "type": "object",
            "additionalProperties": true
          }
        },
        "required": ["source", "target", "type"]
      }
    }
  },
  "required": ["nodes", "relationships"]
}
```

Source Code D.1: The JSON Schema defining the structure for the knowledge graph output.

D.2 Example of a Valid JSON Object

The following listing shows an example of a JSON object that validates against the schema defined in Listing D.1. This demonstrates the format of the data generated by the LLM, containing two nodes (a `Person` and a `University`) and a single relationship (`EDUCATED_AT`) connecting them.

```
{
  "nodes": [
    {
      "id": "person_1",
      "labels": ["Person", "Scientist"],
      "properties": {
        "name": "Marie Curie",
        "birth_year": 1867,
        "nationality": "Polish"
      }
    },
    {
      "id": "uni_1",
      "labels": ["University"],
      "properties": {
        "name": "University of Paris",
        "location": "Paris, France"
      }
    }
  ],
  "relationships": [
    {
      "source": "person_1",
      "target": "uni_1",
      "type": "EDUCATED_AT",
      "properties": {
        "degree": "Doctor of Science",
        "year_graduated": 1903
      }
    }
  ]
}
```

Source Code D.2: An example of a valid knowledge graph JSON object conforming to the schema.

Appendix E: LLM Prompts

This appendix contains a comprehensive list of all prompts used to interact with the large language models (LLMs). Each prompt is presented verbatim, accompanied by a description of its purpose, the context in which it was used, and the expected format of the response. This section is intended to ensure the replicability of the research by detailing the exact inputs given to the models. Note that placeholders, highlighted in red like `@chunk@`, are dynamically populated by the system at runtime.

E.1 Initial Knowledge Graph Extraction

The following prompt is the cornerstone of the data extraction pipeline. It instructs the LLM to act as a knowledge graph expert, analyzing a chunk of text to identify named entities and their relationships. It enforces a strict output format, including the creation of distinct ‘Instance‘ and ‘Type‘ nodes and the mandatory ‘IS_A‘ relationship connecting them.

```
You are an expert assistant that analyzes text to build a knowledge graph. Your goal is to extract
     named entities, their conceptual types, and the relationships between them. Follow the
     instructions and the example precisely.

1. Instructions

A. Instance and Type Nodes: For every named entity you identify:
    Create an Instance Node: This represents the specific entity (e.g., "Ordinance 15-01").
        It must have the single label `["Instance"]`.
        Its `properties` must include:
            `id`: Assign a unique `id` starting with the chunk prefix `cchunk-node-` followed by a
     sequential number (e.g., "cchunk-node-1", "cchunk-node-2").
            `name`: The lowercase version of the entity's text (e.g., "ordinance 15-01").
            `displayName`: The Title Case version of the entity's text (e.g., "Ordinance 15-01").
            `mergeCount`: Initialized to `0`.
            `refinementCount`: Initialized to `0`.
        Add any other properties you find relevant.
    Create a Type Node: This represents the abstract concept (e.g., "Ordinance").
```

```
        It must have the single label `["Type"]`.
        Do not worry about creating duplicate Type nodes; the system will handle them later.
        Its `properties` must include:
            `id`: Assign a unique `id` starting with the chunk prefix `cchunk-node-` followed by a
    sequential number (e.g., "cchunk-node-1", "cchunk-node-2").
            `name`: The lowercase version of the concept's name (e.g., "ordinance").
            `displayName`: The Title Case of the concept's name (e.g., "Ordinance").
            `mergeCount`: Initialized to `0`.
            `refinementCount`: Initialized to `0`.

B. Relationships:
    `IS_A` Relationship: For every Instance Node, you MUST create an `IS_A` relationship connecting
    it to its corresponding Type Node.
    Other Relationships: Create any other relationships you find between Instance Nodes (e.g., `
    AMENDS`, `CONTAINS`).

2. Crucial Example

If the text is "Ordinance 15-01 amends Section 20-7." from chunk chunk, your JSON output must include
    :

An Instance Node for "Ordinance 15-01": {{"id": "cchunk-node-1", "labels": ["Instance"], "properties
    ": {{"name": "ordinance 15-01", "displayName": "Ordinance 15-01", "mergeCount": 0, "
    refinementCount": 0}}}}
An Instance Node for "Section 20-7": {{"id": "cchunk-node-2", "labels": ["Instance"], "properties":
    {{"name": "section 20-7", "displayName": "Section 20-7", "mergeCount": 0, "refinementCount":
    0}}}}
A Type Node for "Ordinance": {{"id": "cchunk-node-3", "labels": ["Type"], "properties": {{"name": "
    ordinance", "displayName": "Ordinance", "mergeCount": 0, "refinementCount": 0}}}}
A Type Node for "DocumentSection": {{"id": "cchunk-node-4", "labels": ["Type"], "properties": {{"name
    ": "documentsection", "displayName": "DocumentSection", "mergeCount": 0, "refinementCount":
    0}}}}
Relationships:
    {{"source": "cchunk-node-1", "target": "cchunk-node-3", "type": "IS_A"}}
    {{"source": "cchunk-node-2", "target": "cchunk-node-4", "type": "IS_A"}}
    {{"source": "cchunk-node-1", "target": "cchunk-node-2", "type": "AMENDS"}}

3. JSON Output

Provide ONLY the JSON output formatted according to the schema. Do not include explanations.

JSON Schema Format:
json_schema

Analyze the following text and follow all instructions, especially the creation of `IS_A`
    relationships for every entity: text_chunk
```

Source Code E.1: Prompt for initial knowledge graph extraction from a text chunk.

E.2 Entity and Relationship Consolidation

After initial extraction, the LTM consolidation process begins. This involves a series of prompts designed to refine the raw graph by merging duplicate nodes.

E.2.1 Merging Instance Nodes

This prompt asks the LLM to perform entity resolution on ‘Instance‘ nodes. It is given a list of nodes and must identify pairs that represent the same real world entity, returning a list of their IDs to be merged.

```
You are a data quality expert responsible for entity resolution. Your task is to identify and merge
     duplicate entity instances from the provided list.
Analyze the following JSON list of entity instances. Identify pairs that represent the exact same
     real world object or concept.

**Key Rules:**
1.  Merge based on semantic similarity or clear redundancy (e.g., "King Theron" and "Theron" refer to
      the same person).
2.  The output **must** be a single, valid JSON object that conforms to the provided schema.
3.  Do **not** include any text or explanations outside of the final JSON object.
4.  Do **not** suggest merging an instance with itself.
5.  If no instances should be merged, return an empty list of pairs.

**JSON Schema:**
json_schema

Provide ONLY the JSON response.

**Entity Instances to Analyze:**
instance_list_json
```

Source Code E.2: Prompt for merging duplicate instance nodes.

E.2.2 Merging Type Nodes

Similar to the instance merging prompt, this prompt focuses on consolidating the conceptual ‘Type‘ nodes. It asks the LLM to identify semantically equivalent types (e.g., ’Regulation’ and ’Rule’) and return their IDs for merging.

```
You are a data quality expert. Analyze the following list of JSON objects, where each object
    represents an entity type with a 'name' and a unique 'id'.
Identify pairs of types that are semantically equivalent (e.g., 'Rule' and 'Regulation' are the same
    concept).

**Key Rules:**
1. The output **must** be a single, valid JSON object that conforms to the provided schema.
2. The `merge_pairs` array should contain pairs of the unique **'id's** for the types that should be
     merged.
3. Do **not** return the names of the types, only their IDs.
4. Do **not** suggest merging a type with itself.
5. If no types should be merged, return an empty list.

**JSON Schema:**
json_schema

Provide ONLY the JSON response.

**Entity Types to Analyze:**
type_list_json
```

Source Code E.3: Prompt for merging duplicate type nodes.

E.3 Ontology and Hierarchy Refinement

These prompts are used to build a more sophisticated ontology by identifying hierarchical relationships and correcting classifications.

E.3.1 Discovering Part-Whole Relationships

Meronymy (part-whole) relationships are inferred using two complementary prompts. The first (`IS_PART_OF`) checks if a given node is a component

of another. The second (HAS_PART) checks if a given node contains other nodes as its parts.

```
You are an ontology expert specializing in meronymy (part-whole relationships).
Your task is to determine if the 'part_candidate' entity is a component of ANY of the 'whole_
    candidates'.

**Instructions:**
1. Analyze the 'part_candidate' (ID and name).
2. Review the list of 'whole_candidates' (ID and name).
3. Identify the SINGLE most likely entity from the list that the candidate is a part of.
4. The relationship must be a clear part-whole connection, either structural (a chapter is part of a
    book) or conceptual (a wheel is part of a car).
5. Return a JSON object with the 'part_id' and the 'whole_id' of the single best match.
6. If no valid part-whole relationship exists, return an empty JSON object: {}.

**JSON Schema:**
json_schema

Provide ONLY the JSON response.
---
**Part Candidate to Analyze:**
part_candidate_json

**List of Potential Wholes:**
whole_candidates_json
---
```

Source Code E.4: Prompt to identify if an entity IS A PART OF another.

E.3.2 Instance Type Correction

This prompt aims to improve the accuracy of the graph's typing. It presents the LLM with an instance node, its current type, and a list of all available types, asking it to suggest a more specific classification if one exists.

```
You are a data quality analyst. Your task is to review an entity instance and its current type, then
    determine if a more specific classification exists from the list of available types.

**Instructions:**
1. Analyze the `instance_name`, `instance_id`, and `current_type`.
2. Review the `available_types`, which is a list of JSON objects, each with a unique `id` and `name
    `.
```

```
3.  If you find a more appropriate type, return a JSON object containing the original `instance_id`
    and the `id` of the new type (`new_type_id`).
4.  If the current type is the most accurate, return an empty JSON object: {}.

**JSON Schema:**
json_schema

**Instance to Review:**
instance_json

**Available Types:**
types_json

Provide ONLY the JSON response.
```

Source Code E.5: Prompt for correcting the type of an instance node.

E.3.3 Organizing Ontology Hierarchy

To create a formal ‘IS_A‘ hierarchy between ‘Type‘ nodes, this prompt provides a child type and a list of potential parent types. The LLM’s task is to select the single most direct parent, establishing a superclass-subclass relationship.

```
Given the child entity type "child_name" (id: "child_id"), which of the following is its most direct
    parent?
A direct parent should be the next most general concept.

**Available parent types (with their IDs):**
potential_parents

**JSON Schema:**
json_schema

Return a single JSON object with the ID of the child and the ID of the selected parent.
```

Source Code E.6: Prompt for organizing the type hierarchy.

E.4 Comparing Knowledge Graphs

For quantitative evaluation, this prompt instructs the LLM to act as an expert system comparing the generated graph against the ground truth. It must count the number of matching entities and relationships and return the results in a structured JSON format.

```
You are an expert system for comparing knowledge graphs. Compare the "Generated Graph" to the "Ground
      Truth" and provide a quantitative analysis.

**Instructions:**
1.  **Analyze Entities**: Count the total entities in the "Ground Truth" and how many of them are
     present in the "Generated Graph". A match occurs if the name is identical or a very close
     semantic equivalent.
2.  **Analyze Relationships**: Count the total relationships in the "Ground Truth" and how many are
     present in the "Generated Graph". A match requires the source, target, and relationship type to
     be correct.

**JSON Schema:**
json_schema

Provide your analysis ONLY in the specified JSON format.
---
**Ground Truth:**
ground_truth_text
---
**Generated Graph:**
generated_graph_text
---
```

Source Code E.7: Prompt for quantitatively comparing two knowledge graphs.

Appendix F: NER Test Documents 1-4: Narrative Baselines

The four short stories (‘NER Test 1‘ through ‘4‘) serve as a contrasting baseline to the legal texts. With higher lexical diversity and more straightforward narrative structures, they allow for rapid testing of the core entity and relation extraction logic in a less ambiguous linguistic environment. Their smaller size also makes them suitable for the manual creation of ground-truth KGs, which are essential for quantitative evaluation of the system’s extraction accuracy. The individual statistical profiles, word clouds, and N-gram analyses for these documents are provided below for completeness.

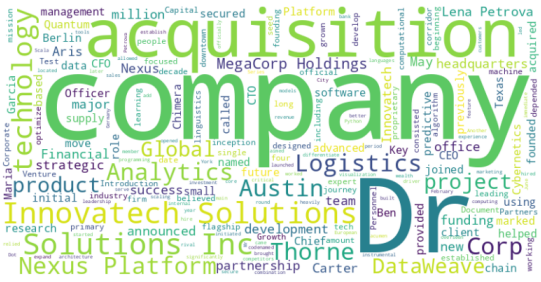


Figure F.1: *Word cloud for NER Test 1.*

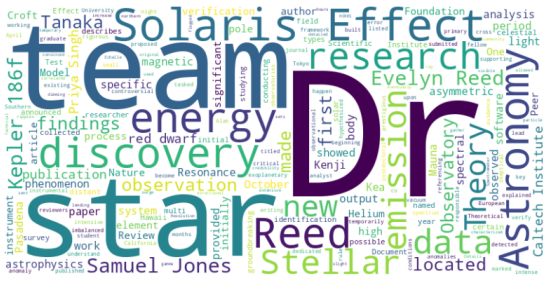


Figure F.2: *Word cloud for NER Test 2.*

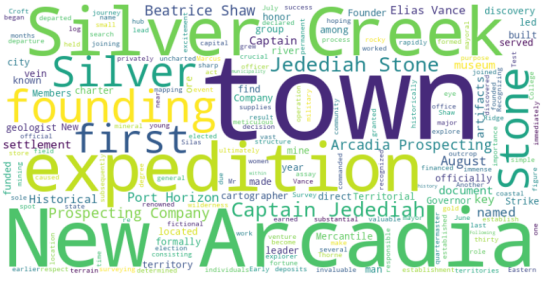


Figure F.3: *Word cloud for NER Test 3.*

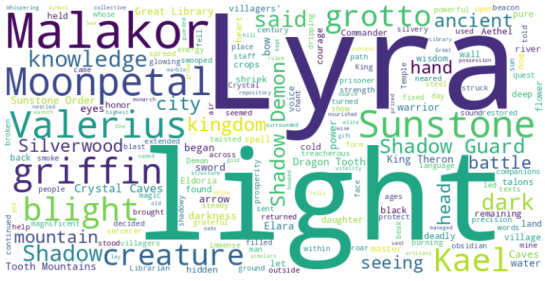


Figure F.4: *Word cloud for NER Test 4.*