Structured Knowledge Extraction from Text using Large Language Models

Master's Thesis of

Mohamad Algoabra

at the Department of Informatics and Electrical Engineering
University of Rostock

Name:	Mohamad Algoabra
Matrikel-Nr.:	219204154
Birthday:	18. October 1998
First Reviewer:	Dr.-Ing. Hannes Grunert
Second reviewer:	Prof. Dr.-Ing. Alke Martens

24. May 2024 – 08. November 2024

Faculty of Computer Science and Electrical Engineering
Institute of Computer Science
University of Rostock
18059 Rostock

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Rostock, 08.11.2024

. . . . . .

(Mohamad Algoabra)

Abstract

This thesis presents an approach to structured knowledge extraction using Large Language Models (LLMs), specifically addressing the challenge of transforming unstructured text into ontology-guided knowledge representations. We introduce a dual-task framework that first generates domain-specific ontologies and then extracts knowledge in the form of custom hypergraphs, ensuring both structural consistency and semantic accuracy. Through the implementation of Parameter-Efficient Fine-Tuning techniques, particularly Low-Rank Adaptation (LoRA), we demonstrate how LLM can be effectively adapted for complex knowledge extraction tasks while modifying less than 1% of the model's parameters.

Our methodology integrates several components: a synthetic data generation pipeline for creating training instances, a validation framework ensuring ontological consistency, and a custom hypergraph representation capable of capturing entities, binary relations, complex multi-entity relations and their attributes. We conducted two distinct sets of experiments – full block adaptation and selective attention-layer adaptation – each tested with different LoRA rank configurations (4, 16, and 32) to investigate how the type of targeted layers and number of adapted parameters affect performance.

The experimental results demonstrate that full-block adaptation achieves superior performance across structural consistency and knowledge similarity metrics, with rank-16 configuration offering an optimal balance between efficiency and effectiveness. Although attention-only adaptation shows promise for computational efficiency by requiring only one-third of the parameters, it exhibits higher volatility in training and lower performance metrics. This research contributes to the field by providing a framework for adapting LLMs to structured knowledge extraction tasks, offering insights into the balance between model efficiency and extraction accuracy, and establishing a foundation for future work in automated knowledge management systems.

Keywords: Knowledge Extraction, Large Language Models, Parameter-Efficient Fine-Tuning, Hypergraphs, Ontology Generation, Natural Language Processing

Acknowledgments

I would like to express my appreciation to my parents for their unwavering support, encouragement, and patience during this period. Their love, understanding, and moral support were indispensable to keeping me motivated and focused on my goals. Their belief in me never wavered, and I am grateful for the sacrifices they made for me.

I would also like to extend my sincere gratitude to my supervisor Dr. Ing. Hannes Grunert, for his invaluable guidance, support, and cooperation throughout my thesis writing journey. His expertise and willingness to share his knowledge have been instrumental in shaping my research and helping me achieve my academicgoal. I am also deeply grateful to Daniel Pokrandt for providing me with the necessary equipment and resources to conduct my experiments, which greatly contributed to my research.

Once again, thank you to my supervisor and my parents for their invaluable contributions to my thesis writing.

Contents

List of Figures

List of Tables

1 Introduction

In the contemporary digital era, an unprecedented volume of data is generated daily, predominantly in unstructured textual formats across numerous platforms such as social media, news articles, and scientific publications. The potential to harness this vast reservoir of knowledge is immense; however, it remains largely untapped due to the inherent complexity and variability of unstructured text. Traditional text analysis tools and techniques struggle to manage the scale and intricacy of such data, frequently failing to extract meaningful information effectively. This untapped potential represents a significant opportunity for advancements in various fields.

The rapid growth of digital content presents both a challenge and an opportunity. In healthcare, for instance, vast amounts of clinical notes, patient records, and research articles are produced daily. Extracting meaningful insights from this data could lead to improved patient care, more accurate diagnoses, and the discovery of novel treatments. In finance, analyzing market trends, news reports, and financial statements could enhance investment strategies and risk management. Public administration could benefit from analyzing social media trends, public opinions, and policy impacts to improve governance and policy-making.

Recent advancements in Artificial Intelligence (AI), particularly the development of transformer based [54] large language models (LLMs) such as BERT [8], GPT [32], and LLaMA [51], have significantly transformed the field of Natural Language Processing (NLP). These models exhibit exceptional proficiency in comprehending and generating human-like text by capturing deep contextual nuances that previous models could not. Unlike other methods, LLMs are trained on extensive textual datasets across various domains using deep learning techniques. This training enables them to understand human language more effectively, resulting in superior performance in tasks such as language translation, sentiment analysis, question answering and summarization.

This thesis is motivated by the potential of these models to fundamentally transform unstructured text into structured, actionable knowledge, thereby unlocking new opportunities for data analysis and decision-making in various fields such as healthcare, finance, and public administration. By converting unstructured text into structured data, we can enhance the ability to perform complex analyses, identify patterns and trends, and make informed decisions. Furthermore, the integration of LLMs into existing data workflows can significantly reduce the manual effort required for data processing and analysis. This automation can lead to more efficient and cost-effective operations across various industries. By leveraging the capabilities of these models, organizations can focus their resources on higher-level strategic activities rather than routine data handling tasks.

1.1 Problem Formulation

Upon recognizing the critical importance of extracting structured knowledge from textual data, it becomes essential to investigate methodologies for adapting LLMs to achieve this objective. This thesis aims to develop such a methodology, focusing on dynamic extraction of entities, relations, and attributes. By identifying entities, the relations among them and their attributes within unstructured text, we provide a systematic framework for converting the intrinsic complexity of unstructured text into structured, actionable knowledge. This approach not only enhances our ability to process textual data but also offers a clear pathway for transforming raw information into valuable insights.

1.1.1 Core Challenges

Despite the advancements in NLP methods, significant challenges persist in the extraction and structuring of knowledge from textual data. Current systems often fall short in such tasks when the complexity of the text increases or when the domain changes. Some of these challenges include:

- **Ambiguity and Polysemy:** Words and phrases frequently exhibit multiple meanings depending on their context. Disambiguating these meanings is crucial for precise information extraction.
- **Domain adaptability:** Adapting to diverse domains and evolving data without extensive retraining or manual intervention. This challenge requires systems that can generalize well across various topics and subject matters.
- **Coreference Resolution:** Resolving references to the same entities within a text, such as pronouns or repeated mentions, is essential for maintaining coherence in the extracted information. Accurate coreference resolution is necessary to link different parts of the text referring to the same entity.
- **Low-Resource Languages:** Many languages suffer from limited available training data, posing challenges for developing robust models.
- **Out-of-Vocabulary Words:** Handling words that were not present in the training data can lead to challenges.

1.1.2 Hypothesis

In light of the advancements and potential of LLMs in NLP, this thesis proposes a hypothesis to explore and validate the extent of their capabilities. By examining the inherent features and performance of LLMs, we aim to understand how these models can be effectively utilized to transform unstructured textual data into structured, actionable knowledge. The hypothesis is divided into several key aspects that highlight the strengths of LLMs.

Our hypothesis is centered around three key aspects:

1. **Domain Knowledge and Adaptability:** We propose that LLMs encapsulate a wide range of domain knowledge due to their extensive pre-training on vast and diverse datasets. This broad exposure enables LLMs to develop a deep understanding of language, facilitating generalization across different topics and contexts. We believe that this intrinsic knowledge allows LLMs to perform high adaptability to various domains straight out of the box, without requiring extensive retraining or fine-tuning.

2. **Contextual Understanding:** We hypothesize that LLMs possess a superior ability to understand and interpret text within its specific context. This includes accurately determining the meaning of words based on their usage, resolving ambiguities, and handling multiple meanings of words and phrases. This capability is essential for practical applications where precise comprehension of nuanced language is required.

3. **Multilingual Capability:** A critical aspect of our hypothesis is the multilingual capability of LLMs. We propose that these models can understand and process multiple languages without requiring extensive retraining for each language. This capability significantly reduces the need for language-specific models, thereby enhancing the versatility and applicability of LLMs in multilingual contexts.

By exploring these hypotheses, we aim to demonstrate the transformative potential of LLMs in NLP, showcasing their ability to handle a wide array of linguistic challenges and applications effectively.

1.1.3 Limitations

Despite the promising capabilities of LLMs, they face significant challenges that need to be addressed to realize their full potential:

1. **Hallucination:** Hallucination refers to the model's tendency to generate content that is not supported by real knowledge or input prompts, often resulting in inaccuracies or misinformation [16]. This phenomenon can undermine the reliability of the information produced by LLMs, especially in critical applications where accuracy is paramount.

2. **Sensitivity to Prompts:** LLMs exhibit sensitivity to the prompts due to their autoregressive nature, where each word is predicted sequentially based on the preceding words, which leads to error overlap and lack of coherence in the extended text, negatively affecting the accuracy of the model [38, 3]. Thus, maintaining control over the output, especially for tasks requiring strict adherence to specific data or formats, becomes challenging.

These challenges highlight the difficulty in ensuring reliable and structured text output from LLMs, necessitating further research and development to mitigate these issues.

1.1.4 Research question

Building upon the motivation, problem formulation, and hypothesis, the primary research question this thesis seeks to address is:

Research question

How can large language models be effectively adapted to extract structured knowledge from unstructured text across various domains?

This question aims to explore the methodologies and frameworks that can harness the inherent capabilities of LLMs for transforming unstructured text into structured, actionable knowledge. The goal is to investigate the strategies that enhance the efficiency, accuracy, and reliability of these models in various contexts.

1.2 Objectives of the Thesis

To address this research question, the thesis aims to leverage the capabilities of transformer-based LLMs to enhance the extraction and structuring of knowledge from unstructured texts. The specific objectives are as follows:

1. Develop and fine-tune LLMs for specific tasks such as Named Entity Recognition (NER) and Relationship Extraction (RE) to improve performance and adaptability across different domains.

2. Evaluate the effectiveness of transformer-based LLMs in extracting structured knowledge from unstructured text sources, focusing on accuracy, efficiency, and scalability.

3. Integrate extracted entities and relationships into coherent structured knowledge representation such as knowledge graphs or relational models that are readily usable for data analysis and decision-making, thereby demonstrating the practical value of structured data extraction in real-world applications.

Through these objectives, the thesis seeks to contribute to the field of NLP by advancing the understanding and application of transformer-based models in structured knowledge extraction, ultimately facilitating more informed and effective decision-making processes across various sectors.

1.3 Structure of the Thesis

This section aims to provide readers with a concise overview of the organization and content of each chapter in this thesis. By presenting a summary of the structure and substance of each chapter, we hope to enhance readers' comprehension of the research material and facilitate easier navigation through it.

Chapter 2 provides the theoretical foundation by introducing key concepts in Natural Language Processing (NLP), explaining essential tasks like Named Entity Recognition and Relation Extraction. It examines the Transformer model architecture in detail, including its components and variations, followed by a comprehensive overview of Language Model training processes and parameter-efficient fine-tuning methods. The chapter concludes by exploring different approaches to structured knowledge representation, comparing relational models, knowledge graphs, and hypergraphs. This theoretical groundwork sets the stage for the research methodology and implementation presented in subsequent chapters.

Chapter 3 presents the literature review, examining recent developments in Large Language Models (LLMs) like GPT-4, LLaMA 3.1, and Mistral models, alongside state-of-the-art approaches in structured knowledge extraction such as REBEL and LLM-NERRE. The review reveals significant advancements in both model capabilities and extraction techniques, while identifying crucial gaps including limited schema validation, inadequate handling of complex relationships, and insufficient support for attribute-rich knowledge representation. This analysis provides the foundation for the research methodology presented in subsequent chapters.

Chapter 4 presents our methodology for extracting structured knowledge from unstructured text through a dual-task approach: first generating an ontology schema from the input text, then extracting a knowledge hypergraph guided by this ontology. Built on Large Language Models and custom hypergraph representations, the methodology details the core components: a conceptual framework integrating LLMs with knowledge representation, a modular system architecture, a synthetic data preparation pipeline, model fine-tuning using LoRA adapters for each task, and evaluation methods. This approach ensures both flexibility in knowledge extraction and strict ontological validation of the extracted information.

Chapter 5 details the technical implementation of our knowledge extraction system, describing the development environment, followed by four key components: a data pipeline that transforms synthetic training data into standardized prompts for dual-task learning, a model fine-tuning implementation using LoRA with 4-bit quantization for efficient adaptation of LLaMA 3.1 8B, an evaluation system that comprehensively assesses both ontology generation and knowledge extraction capabilities, and a knowledge integration module that employs Neo4j for storing and querying the extracted knowledge in a graph database structure supporting both binary relations and complex hyperedge representations.

Chapter 6 presents the experimental evaluation of our knowledge extraction system through two distinct experimental approaches: full block adaptation and selective attention-layer adaptation of LLaMA 3.1 8B, each tested with various LoRA configurations. The chapter examines training dynamics through loss curves and evaluates performance using three key metrics: structural consistency, knowledge similarity, and ontology adherence. Through detailed analysis of both adaptation strategies and their parameter efficiency, the experiments demonstrate the system's capability to perform both ontology generation and knowledge extraction tasks, while highlighting the trade-offs between computational efficiency and model performance across different configurations.

Finally, in **Chapter 7** we present the conclusions drawn from our research, summarizing the effectiveness of using LLMs for structured knowledge extraction through our dual-task approach. The chapter discusses the key limitations of our current implementation and outlines promising directions for future work such as expanding multilingual capabilities, exploring multi-modal knowledge extraction, and enhancing the system's ability to handle more complex ontological relationships. These insights provide a foundation for further advancement in automated knowledge extraction systems.

2 Theoretical Foundation

This chapter provides an overview of the foundational and background concepts that form the basis of the research in this thesis. It discusses the development and current trends in Natural Language Processing, highlighting the significant influence of transformer models. For more in-depth information and additional references, please refer to the cited sources.

2.1 Natural Language Processing

Natural Language Processing (NLP) is a subfield of artificial intelligence that focuses on developing systems capable of processing and analyzing human language in written, spoken, and other formats. The aim of NLP is to enable computers to understand, interpret, and manipulate human language in a way that is both meaningful and useful. This capability is essential in numerous applications, including but not limited to text generation, sentiment analysis, machine translation, named entity recognition and content summarization [7].

The evolution of NLP can be broadly categorized into distinct phases. Early efforts in the mid-20th century relied on symbolic methods, employing hand-coded linguistic rules [6]. However, this approach struggled with the inherent complexity of language. Then a paradigm shift towards statistical and machine learning models emerged in the late 1980s and 1990s. This data-driven approach facilitated the extraction of patterns within language [20, 6]. The introduction of machine learning algorithms further enhanced these techniques.

A breakthrough came with the advent of neural networks, which have fundamentally transformed the landscape of NLP since the early 2010s. Although Recurrent Neural Networks (RNN) [36] and Long Short-Term Memory networks (LSTM) [11] were developed in the 1980s and 1990s, their potential wasn't fully realized due to limited computational resources at the time. It wasn't until more advanced hardware and algorithms became available that these models allowed for better handling of sequential data, such as sentences in text.

The real paradigm shift occurred with the introduction of the transformer model in 2017 [54]. This model discarded the recurrent layers in favor of attention mechanisms, which enable the model to focus on different parts of the text as needed without the constraints imposed by the sequential nature of RNNs. Transformers have since become the backbone of modern NLP, with models like BERT [8] and GPT [32], which have set new standards for a variety of NLP tasks due to their deep contextual understanding and generation capabilities.

2.1.1 Tokenization and Numerical Encoding

The preprocessing of textual data for NLP tasks and training LLMs is important and involves several steps that transform raw text into a format suitable for machine understanding and analysis.

Tokenization

Tokenization is the initial step in NLP [55], where text is segmented into smaller units known as tokens, which can be words, subwords, or characters depending on the task, to form a vocabulary from a unique set of these tokens [29]. Modern NLP practices often use subword tokenization methods like Byte-Pair Encoding [39] or WordPiece [57]. These methods strike a balance between vocabulary size and the ability to decompose unfamiliar words, thereby reducing model complexity and improving generalization.

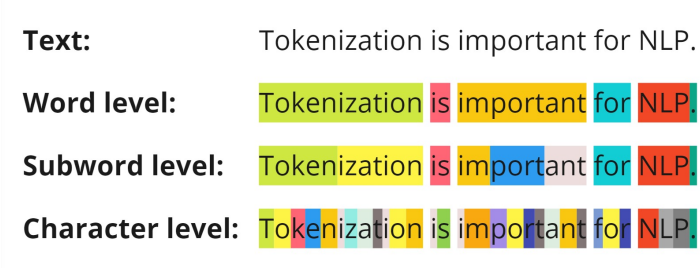


Figure 2.1: Tokenization types

Numerical Encoding

After tokenization, the next step is numerical encoding. In this phase, the tokens are converted into numerical representations that can be processed by neural networks or other machine learning models [30]. Since these models require numerical inputs to function, each token is typically mapped to a unique integer based on its position in the vocabulary. There are several methods of numerical encoding:

- Integer Encoding: Each token is assigned a unique integer. This method is straightforward and efficient but does not capture semantic relationships between tokens.
- One-Hot Encoding: Each token is represented as a binary vector where only the index corresponding to the token is set to 1, and all other indices are set to 0.
- Word Embeddings: Word embeddings map tokens to dense vectors of real numbers in a continuous vector space. These vectors capture semantic relationships between words, where similar words have similar vectors.

These encoding methods convert text tokens into numerical formats suitable for machine learning models, enabling effective NLP applications. Integer and one-hot encoding are simpler and faster, but word embeddings provide richer, more meaningful representations of text, which are crucial for sophisticated NLP tasks [30].

2.1.2 Named Entity Recognition

Named Entity Recognition (NER) is a essential information extraction task in NLP aimed at identifying and classifying key information elements in plain text into predefined categories [53, 49]. These elements, referred to as entities, can be both tangible and abstract objects, and their specific definitions can vary across different domains.

An entity can be represented by a single token or a multi-token expression and are categorized by their respective entity types. The process of NER is divided into two primary steps:

1. **Entity Detection:** This step involves identifying sequences of one or more tokens that constitute an entity, without yet specifying the entity type.
2. **Entity Classification:** In this step, the detected entities are assigned to specific categories.

However, most NER methods simultaneously perform both steps without making an explicit distinction between them.

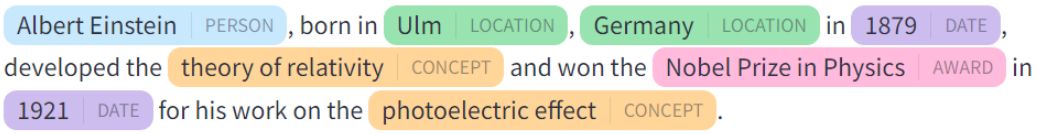


Figure 2.2: Named Entity Recognition example

As illustrated in Figure 2.2, NER can identify a variety of entity types within a text. For instance, single words like "Ulm" and "Germany" are classified as LOCATION, while multi-word expressions such as "Albert Einstein" and "theory of relativity" are labeled as PERSON and CONCEPT, respectively. This example underscores the versatility of NER in detecting a diverse range of entities, including names, locations, dates, concepts, and awards.

2.1.3 Named Entity Normalization

Named Entity Normalization (NEN), also known as entity linking, is an NLP task that aims to transform different mentions of the same entity into a single, standardized form [21, 40]. This process is essential for ensuring consistency and accuracy in extracted data, particularly in applications such as information retrieval, text mining, and knowledge base construction. By normalizing entities, we can unify various forms of referring expressions to the same object, thereby enhancing the quality of downstream tasks and facilitating more effective information retrieval and analysis.

NEN addresses the challenge of how entities are referred to in text, where entities, such as names of people, organizations, locations, and dates, frequently appear in multiple forms within and across documents. These variations can arise due to differences in spelling,

abbreviations, aliases, or even typographical errors. For instance, a person's name might be written as "Albert Einstein" in one document and "Prof. Einstein" in another. Similarly, a location might be referred to as "New York City" in one context and "NYC" in another. These variations can lead to inconsistencies and inaccuracies in extracted data, making it challenging to analyze and draw meaningful insights from the data.

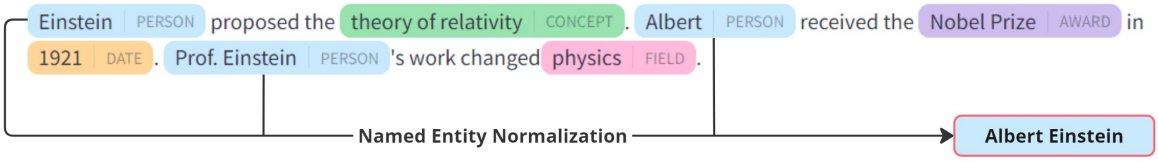


Figure 2.3: Named Entity Normalization example

As illustrated in Figure 2.3, NEN tackles this issue by standardizing all mentions of an entity into a single, unambiguous form. In the example, mentions such as "Einstein," "Albert," and "Prof. Einstein" are all normalized to "Albert Einstein." This ensures that all variations refer to the same entity consistently across the text.

2.1.4 Relation Extraction

Relation Extraction (RE) is another NLP task that build upon NER. While NER focuses on identifying and classifying individual entities within text, RE delves deeper into the semantic structure of text by identifying the relationships between the identified entities. This shift from entity recognition to relationship extraction enriches the understanding of the conveyed information and allows for a more comprehensive analysis of textual data.

RE methods primarily focus on two key aspects:

1. **Entity Pair Detection:** This stage identifies the specific entities within a sentence that are engaged in the relation. This step often follows NER, where potential entity pairs are generated based on their co-occurrence in sentences or documents.

2. **Relation Classification:** Once the entities are identified, the system classifies the relationship type that governs the interaction between them. Common relationship types encompass "located in," "works at," "married to," or "cause-effect" (as exemplified in the aforementioned sentence).

However, modern RE methods jointly learn to identify entities and their semantic relations without making an explicit distinction between them [26].

As illustrated in Figure 2.4, the sentence: "Albert Einstein won the Nobel Prize in Physics for his work on the photoelectric effect." NER would successfully locate "Albert Einstein" as a PERSON entity, "Nobel Prize in Physics" as an AWARD entity, and "photoelectric effect" as a CONCEPT entity. RE, however, progresses further by recognizing the relationship

between these entities, which in this instance is RECEIVED for the connection between "Albert Einstein" and "Nobel Prize in Physics," and WORKED_ON for the connection between "Albert Einstein" and "photoelectric effect." These extracted relationships provide a detailed understanding of the achievements of "Albert Einstein" and the context of his award.

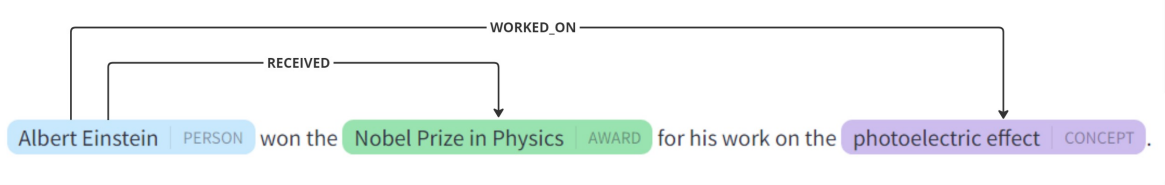


Figure 2.4: Relation Extraction example

The importance of RE extends to various applications, including knowledge base construction, question answering, and text summarization. By extracting relationships, we can construct structured representations of unstructured text, enabling machines to understand and utilize the information more effectively.

2.2 Transformer Model

The NLP field has seen significant advancements over the past decades, largely driven by the development of neural network models. Early architectures such as RNNs (Recurrent Neural Networks) and LSTMs (Long Short-Term Memory networks) were groundbreaking in their ability to handle sequential data. However, these models struggled with limitations like difficulty in capturing long-range dependencies and high computational complexity due to their sequential nature. These challenges necessitated a new approach, paving the way for the development of the Transformer model.

In 2017, Vaswani et al. introduced the Transformer model in their seminal paper "Attention is All You Need" [54]. This model represented a paradigm shift in NLP by eliminating the need for recurrent structures and instead relying entirely on attention mechanisms. The core idea was to use self-attention to process input data in parallel, significantly improving computational efficiency while simultaneously mitigating the vanishing gradient problem in long sequences.

Additionally, the Transformer's parallel data processing capability significantly enhances training efficiency. Unlike the recurrent structures, which process sequences sequentially, Transformers handle entire sequences simultaneously. This parallelism allows for the training of larger models, leading to the development of large language models or LLMs.

2.2.1 Architecture Details

The Transformer model is fundamentally structured around an encoder-decoder architecture, where both the encoder and the decoder are built from a stack of identical layers.

The encoder is primarily responsible for transforming the input sequence into a continuous representation called an embedding. It consists of a series of N identical layers, each containing key components such as multi-head self-attention mechanisms, position-wise feed-forward networks, and residual connections with layer normalization. These components work together to capture contextual information from the input sequence and create rich, continuous representations.

Following the encoder, the decoder generates the output sequence based on both its previous outputs and the encoder's representations. Like the encoder, the decoder is composed of N identical layers. Each layer includes a masked multi-head self-attention mechanism, which ensures that predictions at a given position depend only on previously generated tokens, in addition to the standard multi-head self-attention mechanism, position-wise feed-forward networks, and residual connections with layer normalization. These mechanisms allow the decoder to incorporate context from both the encoder and its own prior outputs to produce coherent and accurate predictions.

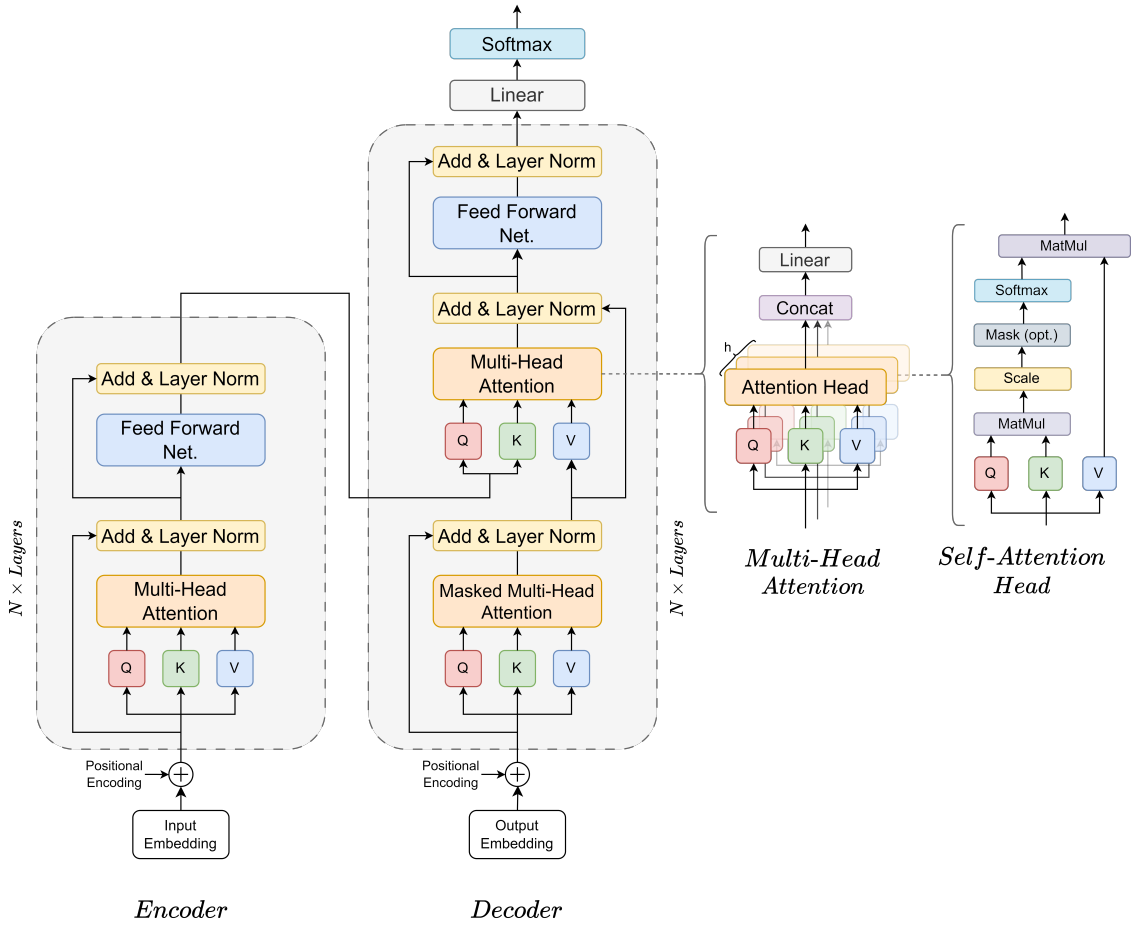
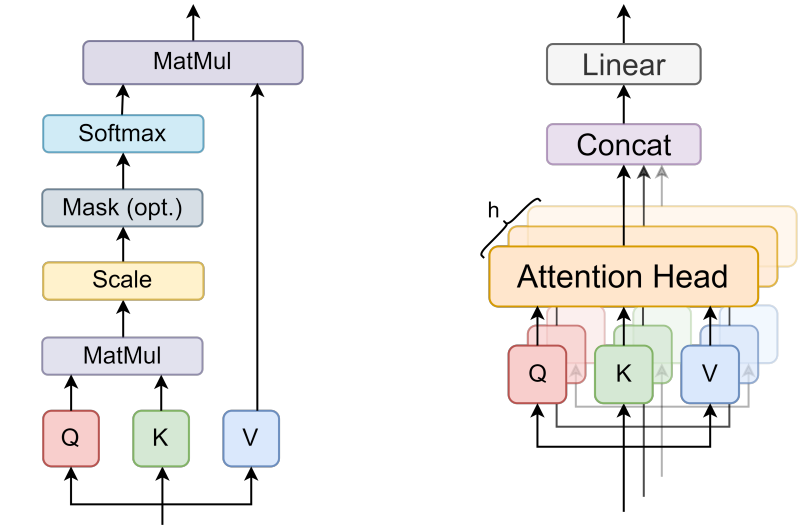


Figure 2.5: The Transformer – Model architecture

As illustrated in Figure 2.5, the encoder and decoder are composed of several building blocks. These blocks include:

Self-Attention Mechanism

At the core of the Transformer architecture is the self-attention mechanism, which enables the model to weigh the influence of different words in a sentence when encoding a single word. This mechanism is essential for capturing the contextual relationships between words, allowing the model to focus on the most relevant parts of the input sequence.



(a) *Self-Attention Head* (b) *Multi-Head Attention*

Figure 2.6: The Transformer – Self-Attention Head – Multi-Head Attention

As shown in Figure 2.6, the Self-Attention mechanism consists of two main concepts which are the Self-Attention Head and Multi-Head Attention.

- **Self-Attention Head**

  In the Self-Attention mechanism, each word in the input sequence is transformed into three distinct vectors through learned linear transformations, as shown in Figure 2.7. These vectors are abstractions inspired by information retrieval systems and databases that are essential for calculating the attention scores and are defined as follows [1, 31, 52]:

  1. **Query vector Q:** This vector represents the current token we are focusing on. It is used to query the relevance of other tokens in the sequence.
  2. **Key vector K:** This vector represents the tokens we are comparing against. It is used in conjunction with the query vector to determine the attention scores.
  3. **Value vector V:** This vector represents the actual values of the tokens that will be combined based on the attention scores.

Formally, for each word embedding x_i, these vectors are computed as:

$$Q_i = W_Q x_i, \quad K_i = W_K x_i, \quad V_i = W_V x_i$$

where W_Q, W_K and W_V are learned weight matrices. These transformations allow the model to project the input embeddings into different vector spaces, each suited for a specific aspect of the attention mechanism.

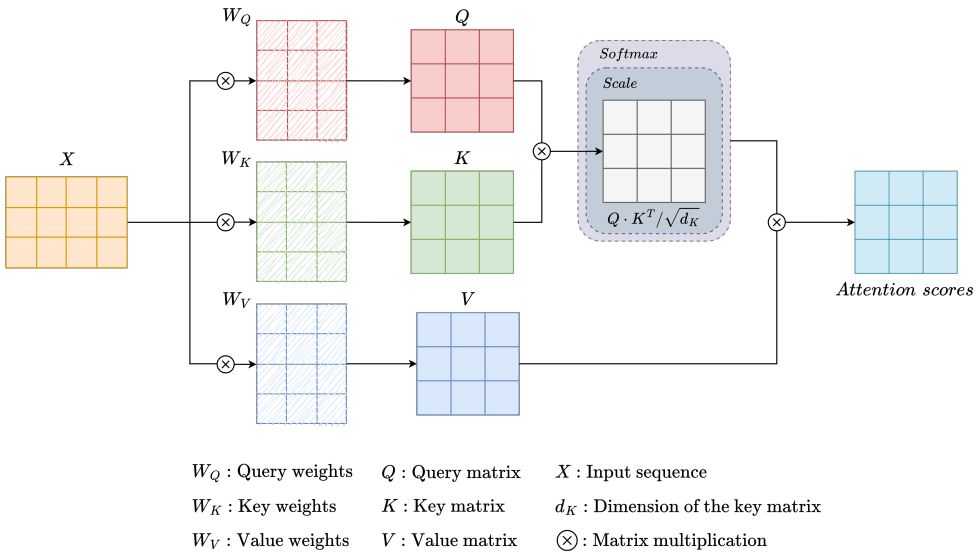


Figure 2.7: The Transformer – Self-Attention Head

The self-attention mechanism calculates attention scores for each word relative to every other word in the sequence using a scaled dot-product method, as described by the following equation:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) V \tag{2.1}$$

This equation can be broken down into the following steps:

1. Dot product of Query and Key vectors: The attention score between a pair of words is computed using the dot product of their Query and Key vectors. This step measures the similarity between the Query and Key vectors.
2. Scaling by the square root of the Key dimension: To prevent the dot products from growing too large, especially when the dimensionality of the vectors is high, the raw scores are scaled by the square root of the key vector's dimension d_K.
3. Applying the softmax function: The scaled scores are then passed through a softmax function to produce the attention weights. This ensures that the weights are normalized, summing up to one.

4. Combining with Value vectors: The attention weights are used to create a weighted combination of the value vectors. This results in a context vector that captures the relevant information from the entire sequence for the given query word.

Through this process, the self-attention mechanism effectively allows the model to focus on different parts of the input sequence, dynamically adjusting its attention based on the context provided by the entire sequence. This ability to capture contextual relationships is what makes the self-attention mechanism a powerful tool in natural language processing tasks.

Additionally, in the decoder, a variant of the self-attention mechanism known as masked self-attention is used, see Figure 2.5. This mechanism ensures that the prediction for a given position depends only on the previously generated tokens and not on future tokens. The key difference is that during the computation of attention scores, positions corresponding to future tokens are masked out (set to a very negative value before applying the softmax function), ensuring that the model cannot look ahead. This masking is crucial for autoregressive generation tasks, where the model generates the sequence one token at a time.

- **Multi-Head Attention**

  To capture diverse types of relationships between tokens, the Transformer extends the self-attention mechanism with multi-head attention, as shown in Figure 2.6. This approach allows the model to focus on different aspects of the input sequence simultaneously, providing a richer representation of the data.

  Instead of performing a single attention function, the multi-head attention mechanism employs multiple attention heads. Each head operates independently with its own set of query, key, and value matrices, enabling the model to capture different features of the input sequence. The steps involved in multi-head attention are as follows:

  1. **Linear Projections:** For each attention head, the input embeddings are linearly projected into multiple sets of Query, Key, and Value vectors. These projections are performed using different learned weight matrices for each head.
  2. **Parallel Attention:** Each attention head independently computes the attention scores and context vectors using the self-attention mechanism described earlier. This allows each head to focus on different parts of the input sequence and capture various aspects of the relationships between tokens.
  3. **Concatenation of Heads:** The context vectors from all attention heads are concatenated to form a single combined vector. This step integrates the diverse information captured by each head.
  4. **Linear Transformation:** The concatenated vector is then linearly transformed using another learned weight matrix to produce the final output. This transformation ensures that the output has the same dimensionality as the original input embeddings.

The multi-head attention mechanism can be mathematically described by the following equation:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O \tag{2.2}$$

where each head is computed using equation 2.1 as follows:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Here, W_i^Q, W_i^K, and W_i^V are the learned weight matrices for the i-th attention head, and W^O is the learned weight matrix for the output projection [54].

Positional Encoding

One limitation of the self-attention mechanism is that it does not inherently capture the order of the words in the input sequence. To address this, the Transformer uses positional encoding to provide the model with information about the position of each word.

The positional encoding vectors are element-wise added to the input embeddings to incorporate positional information. These vectors are generated using *sin* and *cosine* functions of different frequencies. The positional encoding for a position *pos* and dimension i is defined as:

$$\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{n^{2i/d_{\text{model}}}}\right)$$

$$\text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{n^{2i/d_{\text{model}}}}\right)$$

Where *pos* represents the position of the token in the sequence, i is the dimension index of the token, d_{model} is the number of dimensions of the input embeddings and n is a constant scaling factor set to 10,000 by the authors to ensure a broad range of frequencies in the encoding [54].

The rationale behind using sine and cosine functions is that they provide the model with a way to learn relative positions [54]. When these positional encoding vectors are added to the input embeddings, they help maintain meaningful distances between embedding vectors. This positional information is preserved when the embeddings are projected into Query, Key, and Value vectors during the dot-product attention process, enhancing its ability to understand and generate coherent text.

Position-wise Feed-Forward Network

The feed-forward network in the encoder and decoder parts of the Transformer is a simple two-layer fully connected neural network, but with a slight modification. Instead of processing the whole sequence of embeddings as a single vector, it processes each embedding independently. This design is often referred to as a position-wise feed-forward network [54]. In the position-wise feed-forward network, the transformations are applied independently to each position in the sequence. This means that for each token x_i in the

input sequence, the feed-forward network processes x_i without considering other tokens in the sequence.

If we denote the input sequence by $[x_1, x_2, \ldots, x_n]$, where n is the sequence length, the position-wise feed-forward network applies the same operations to each x_i individually:

$$\text{FFN}(x_i) = \max(0, x_i W_1 + b_1) W_2 + b_2 \quad \text{for each } i \in [1, n] \tag{2.3}$$

This equation can be broken down into the following steps:

1. **None-Linear Transformation:** The input x_i is first transformed by a learned weight matrix W_1 and bias b_1, followed by a ReLU activation function. This introduces non-linearity to the model, allowing it to learn more complex representations.

$$\text{Intermediate}_i = \max(0, x_i W_1 + b_1)$$

2. **Linear Transformation:** The intermediate result is then transformed by a second learned weight matrix W_2 and bias b_2 to produce the final output.

$$\text{Output}_i = \text{Intermediate}_i W_2 + b_2$$

By applying the feed-forward network position-wise, each embedding is processed independently, ensuring that the positional information of each word is preserved. This approach maintains the integrity of each word's positional context within the sequence while still benefiting from the non-linear transformations provided by the feed-forward layers. This design allows the Transformer to effectively learn and represent the input data, capturing both the positional and contextual relationships between tokens.

Layer Normalization and Residual Connections

As mentioned earlier, the Transformer architecture uses layer normalization and skip connections within both the encoder and decoder components, see Figure 2.5. These mechanisms are used to stabilize the training process, improve convergence, and enable the model to learn more effectively from the data. By addressing common issues like internal covariate shift and vanishing gradients.

- **Layer Normalization**

  Layer normalization is a normalization technique introduced in [2] that was applied to the inputs of each sub-layer (i.e., self-attention and feed-forward layers) within the Transformer. This technique normalizes the input across the features, ensuring that the mean and variance are consistent across different layers. The layer normalization is mathematically defined as:

$$\text{LayerNorm}(x_i^l) = \frac{x_i^l - \mu^l}{\sigma^l} \cdot \gamma + \beta \tag{2.4}$$

with:

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} x_i^l$$

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(x_i^l - \mu^l)}$$

where x_i^l is the input at the i-th position of layer l, H is the number of features (hidden units) in the layer, μ^l is the mean, σ^l is the standard deviation, and γ and β are learned parameters that allow the layer to scale and shift the normalized values.

- **Residual Connections**

  Residual connections are skip connections that learn residual functions with reference to the layer inputs, instead of learning unreferenced functions to address the vanishing gradient problem and enable the training of deeper networks [10]. In the Transformer, a residual connection is added around each sub-layer, followed by layer normalization. The output of each sub-layer can be represented as:

$$\text{Output} = \text{LayerNorm}(x + \text{Sublayer}(x))$$

  Here, x is the input to the sub-layer, and $\text{Sublayer}(x)$ represents the transformation applied by the sub-layer (i.e., self-attention or feed-forward layer). The addition of the input x to the output of the sub-layer helps preserve the original information and provides a direct path for the gradient to flow during backpropagation, which mitigate the vanishing gradient problem.

2.2.2 Variations of Transformer Architectures

While the Transformer model was originally introduced as an encoder-decoder architecture (see section 2.2.1), not all LLMs use this design. There are different architectural variations, including encoder-only and decoder-only Transformers, each tailored for specific types of tasks and applications.

Encoder-based transformer

Encoder-only Transformers, such as BERT like model [8], are primarily designed for tasks that involve understanding and interpretation of text, where at each stage, the attention layers can access all the words in the sentence. These models are characterized by:

- **Bidirectional Contextualization:** They can process input in both directions (left-to-right and right-to-left), allowing them to capture context from the entire input sequence simultaneously [8]. This allows the model to understand the full context of each word in a sentence, which is crucial for tasks like question answering, sentiment analysis, and token classification.

- **Masked Language Modeling:** The pre-training objective for encoder-only models involves masked language modeling. In this approach, random tokens in the input are masked during training, and the model learns to predict these masked tokens based on the surrounding context [8]. This objective is formally defined as:

$$\mathcal{L} = - \sum_{i \in \text{masked positions}} \log P(x_i | x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \tag{2.5}$$

  Where, $\mathcal{L}$ represents the loss function, and $P(x_i \mid x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ denotes the probability of the masked word x_i given the surrounding context. The goal is to maximize this probability, thereby minimizing the loss. This process enables the model to develop a deep understanding of context and semantics, crucial for various natural language understanding tasks.

Decoder-based transformer

Decoder-only Transformers, such as GPT models [32], focus on generative tasks. These models are designed to generate text sequences by predicting the next word in a sequence, making them highly effective for tasks like text completion, language modeling, and dialogue generation. These models are characterized by:

- **Unidirectional Contextualization:** Unlike encoder-based models, decoder-only Transformers process input sequentially from left to right, with each token only attending to previous tokens in the sequence [32, 33]. This means that each token can only attend to previous tokens in the sequence, not future ones. This approach aligns with the autoregressive nature of text generation, where each token is generated based on the previously generated tokens.

- **Causal Language Modeling:** For decoder-only models, the pre-training objective is causal language modeling, also referred to as next-token prediction. In this method, the attention mechanism is masked to prevent the model from considering future tokens. This ensures the model learns to predict the next token in a sequence using only the preceding tokens. This objective can be mathematically represented as:

$$\mathcal{L} = - \sum_{i=1}^{n} \log P(x_i | x_1, \ldots, x_{i-1}) \tag{2.6}$$

  where $P(x_i \mid x_1, \ldots, x_{i-1})$ denotes the probability of the next token x_i given all preceding tokens in the sequence. The goal is to maximize this probability, thereby minimizing the loss. This training method enables the model to generate fluent and contextually relevant text, making it suitable for various generative applications.

2.2.3 LLM Quantization

Quantization is a technique used to reduce the computational complexity and memory footprint of deep learning models, making it particularly beneficial for LLMs while maintaining their performance as possible. This process involves converting the continuous

values of model parameters, typically represented in high-precision formats like 32-bit floating point, into lower-precision formats, such as 16-bit, 8-bit or even 4-bit integers [22]. By doing so, quantization reduces model size and accelerates inference, enabling the deployment of LLMs on devices with limited resources without significant performance degradation.

2.3 LLM Training Process

The training of LLMs is a complex, multi-stage procedure designed to create models capable of understanding and generating human-like text. This section outlines the three critical phases in LLM training: Pre-Training, Instruction Fine-Tuning, and Preference Alignment. Each phase plays a crucial role in developing models that are not only knowledgeable but also adaptable to specific tasks and aligned with user preferences.

2.3.1 Pre-Training Phase:

The Pre-Training phase is the foundational step in the training process of LLMs. During this phase, models are exposed to vast and diverse text corpora, typically sourced from the internet, books, academic papers, and other extensive unstructured text repositories. The primary goal of pre-training is to enable the model to learn the intricacies of language, such as grammar, syntax, semantics, and a broad understanding of the world [14].

In this phase, the model employs self-supervised learning[1] method, where it predicts missing pieces of text within a given context, facilitating an understanding of language structure and semantics. The specific methodology varies based on the model architecture:

- **Encoder-based models:** Use Masked Language Modeling, where certain words in a sentence are masked, and the model learns to predict these masked words based on the surrounding context.
- **Decoder-based models:** Use Causal Language Modeling, where the model predicts the next word in a sequence, learning to generate coherent text incrementally.

For a more detailed discussion of these architectures, refer to Section 2.2.2. This stage leverages large-scale, unannotated datasets, allowing the model to learn patterns and relationships within the language autonomously. For example, the LLaMA 3 model series has been trained on an extensive dataset comprising 15 trillion tokens sourced from various text corpora[2]. This comprehensive training ensures that the model can handle a wide range of linguistic tasks with high proficiency.

However, the computational costs associated with the pre-training phase are substantial. Training LLMs requires significant computational resources, including powerful GPUs

[1] Self-supervised learning is a method where models learn to generate labels from the input data itself, enabling them to train on large amounts of unlabeled data by solving pretext tasks.

[2] https://ai.meta.com/blog/meta-llama-3/

or TPUs and extensive parallel processing capabilities. The process can take weeks or even months to complete, depending on the size of the dataset and the complexity of the model. The energy consumption and financial costs are also considerable, often requiring specialized infrastructure and significant investment.

2.3.2 Instruction Fine-Tuning Phase:

The instruction fine-tuning phase represents an important step in the training of task-oriented language models. This phase builds upon the foundational knowledge acquired during the pre-training phase, aiming to enhance the model's ability to interpret and execute specific instructions across a diverse range of tasks [28].

The theoretical basis for instruction fine-tuning lies in transfer learning[3] and domain adaptation principles. It leverages the broad language understanding developed during pre-training phase and adapts it to specific task domains by updating the model's parameters through supervised learning[4] on instruction-output pairs dataset [28, 56]. This process can be formalized as:

$$\theta_{ft} = \arg\min_{\theta} \mathcal{L}(\mathcal{D}_{inst}, f\theta(x)) \tag{2.7}$$

where θ_{ft} represents the fine-tuned model parameters, $\mathcal{D}_{inst}$ is the instruction-output dataset, $f\theta(x)$ is the model's output given input x, and $\mathcal{L}$ is the loss function, typically cross-entropy, defined as:

$$\mathcal{L} = -\sum_{(x,y)\in\mathcal{D}_{inst}} \log P(y|x;\theta) \tag{2.8}$$

where (x, y) are instruction-output pairs from the dataset $\mathcal{D}_{inst}$. This dataset usually consists of pairs for a single task or for a wide range of tasks in the case of multi-task learning, covering a wide range of tasks such as question answering, summarization, translation, and creative writing. This diversity ensures that the model can adapt to various task formats and requirements.

The computational costs during the fine-tuning phase are lower compared to the pre-training phase but still significant. Fine-tuning requires substantial computational resources to process the annotated datasets and iteratively adjust the model's parameters. Depending on the complexity of the tasks and the size of the model, this phase can take several days to weeks.

Overall, the instruction fine-tuning phase bridges the gap between general language understanding and task-specific performance, ensuring that the model is not only knowledgeable but also practically useful in real-world applications.

[3] Transfer Learning is a method where a model trained on one task is adapted for a related task, leveraging pre-existing knowledge to address new problems with limited data.

[4] Supervised Learning is a method where models are trained on labeled data, learning to map inputs to outputs based on example input-output pairs to make predictions on new, unseen data.

2.3.3 Preference Alignment Phase:

The Preference Alignment Phase typically constitutes the final stage in the training process of language models, aiming to ensure that the model's outputs align with human preferences, ethical considerations, and specific use-case requirements. Simply increasing the size of language models does not inherently enhance their ability to follow a user's intent; indeed, larger models can still produce outputs that are untruthful, toxic, or unhelpful, indicating a misalignment with user expectations [28]. This phase addresses these issues by integrating human feedback and preferences directly into the model's decision-making process.

Reinforcement Learning from Human Feedback

Reinforcement Learning from Human Feedback (RLHF) is a method that uses human feedback to guide the model's learning process. Unlike traditional reinforcement learning[5] methods that rely solely on pre-defined reward functions, RLHF introduces a more dynamic and intuitive approach by integrating human preferences and judgments directly into the training process [4]. This integration allows models to better understand and adapt to complex, nuanced tasks that are often challenging to quantify with conventional reward structures. The RLHF approach typically involves three stages:

1. Data Collection: Human annotators meticulously evaluate the quality of model outputs across various dimensions such as helpfulness, truthfulness, and safety. This feedback can take various forms, such as rankings, ratings, or direct corrections, and is used to create a dataset that reflects human preferences and expectations.

2. Training a reward model: The collected human feedback is then utilized to train a reward model. This model predicts the quality of the outputs based on the human feedback, effectively quantifying how well the model's outputs align with human preferences. The reward model serves as a proxy for human judgment in the next stage.

3. Reinforcement Learning: In the final stage, the reward model is used to fine-tune the original model through reinforcement learning. The model's parameters are adjusted to maximize the predicted reward, thus aligning its behavior more closely with human preferences. This iterative process continues until the model achieves satisfactory performance as judged by the reward model.

RLHF has been successfully implemented in several prominent language models, including ChatGPT and LLaMA models, demonstrating significant enhancements in alignment with human preferences and overall output quality [28, 51].

[5] Reinforcement learning is a type of machine learning where an agent learns to make decisions by performing actions in an environment to maximize the rewards.

Proximal Policy Optimization

Proximal Policy Optimization (PPO) is an reinforcement learning algorithm [37] that is well-suited for preference alignment tasks in language models. PPO balances between simplicity and performance, making it a frequently used in RLHF process to align LLMs [28]. The core idea of PPO is to ensure that the policy (in this case, the language model) updates during training do not deviate excessively from the current policy, thus maintaining stability while still allowing for significant improvements.

PPO achieves this by optimizing a clipped objective function[6] that penalizes changes in the policy that are too large, effectively controlling the step size of updates. This ensures that the learning process remains stable and efficient, even when the reward model, based on human preferences, is complex and highly variable [37]. By applying PPO in the context of RLHF, language models can be fine-tuned to produce responses that better align with human values and ethical standards, enhancing the overall user experience.

Direct Preference Optimization

Direct Preference Optimization (DPO) is a method designed to streamline the alignment of language models with human preferences by directly optimizing for these preferences during the training process. Unlike RLHF, which relies on a separate reward model to guide learning, DPO directly optimizes the model to match human preferences without the need for a separate reward model. This simplification reduces the overall complexity of the alignment process.

In DPO, human feedback is used to create a loss function that directly measures the discrepancy between the model's outputs and the desired outputs as indicated by human evaluators [34]. This loss function is then minimized during the training process, leading to a model that produces outputs more closely aligned with human preferences. DPO can be particularly effective for tasks where human judgments are more subjective and harder to quantify, as it allows for a more direct and nuanced integration of human feedback into the model's training process.

Odds Ratio Preference Optimization (ORPO)

Odds Ratio Preference Optimization (ORPO) is an advanced algorithm designed for monolithic preference alignment without the need for a reference model. This approach is distinct in that it integrates preference optimization directly into the supervised fine-tuning (SFT) process, thus eliminating the need for a separate alignment phase [12].

ORPO integrates preference optimization into the training process by incorporating an odds ratio-based penalty into the loss function, which adjusts the model's outputs based on the likelihood of generating favored versus disfavored responses.

[6] Clipped objective function is an objective function that prevent large updates to the policy by clipping the probability ratio between the new and old policies, thereby ensuring stable and reliable learning.

This approach simplifies the alignment process, reduces computational overhead, and has demonstrated significant improvements in model performance and alignment with human preferences across various benchmarks [12].

2.4 Parameter-efficient fine-tuning

As we have seen in Section 2.3, the process of training and fine-tuning large language models (LLMs) is resource-intensive and expensive. To mitigate these challenges and make the adaptation of LLMs more viable, various parameter-efficient fine-tuning (PEFT) methods have been developed. These methods aim to adapt models to specific tasks by tuning a smaller number of parameters, thus making the process more efficient and cost-effective.

PEFT techniques generally fall into three broad categories:

- Additive methods
- Selective methods
- Reparameterization methods

Each of these approaches optimizes model performance on specific tasks while minimizing the computational and memory overhead associated with training.

2.4.1 Additive Methods

Additive Methods are characterized by the augmentation of the existing model architecture with additional parameters or layers that are specifically introduced for fine-tuning purposes. Instead of updating the entire model, only these newly added components are trained, which significantly reduces the number of parameters that need to be adjusted. This approach can be further divided into two subcategories: Adapter-Based Methods and Prompt-Based Methods.

Adapter-Based Methods

Adapter-Based methods introduce small, trainable modules, commonly referred to as "adapters" into the architecture of the LLM. These adapters are inserted within existing layers of the model, typically after each sub-layer in a Transformer model. The idea behind this approach is to keep the majority of the original model's parameters frozen, while the adapters, which contain only a fraction of the total parameters, are fine-tuned to adapt the model to new tasks.

A prominent example of this approach is the Adapter method introduced in [13]. In this method, small fully connected layers are added after each sub-layer of a Transformer model. These adapters are fine-tuned while the rest of the model parameters remain unchanged. This method is particularly efficient as it requires updating less than 4% of the model's parameters, leading to substantial computational savings while maintaining or even improving the model's performance on specific tasks.

The structure of a typical adapter module consists of:

1. A down-projection layer that reduces the dimensionality of the input.
2. A non-linear activation function (e.g., ReLU).
3. An up-projection layer that restores the original dimensionality.

Prompt-Based Methods

Prompt-based methods represent another form of additive technique where the model is fine-tuned by incorporating prompts directly into the input data or through modifications to the model architecture. These methods include both hard prompts, which are fixed, and soft prompts, which are trainable and optimized during fine-tuning.

Prefix Tuning, introduced in [24], is an example of a prompt-based method where a sequence of trainable virtual tokens, known as a "prefix," is prepended to the input sequence. During fine-tuning, only this prefix is optimized, while the original model parameters are largely kept intact. This method enables the model to be adapted to specific tasks with minimal adjustments to the model's overall parameter set, making it an efficient and flexible approach to fine-tuning.

The prefix tuning process can be summarized as follows:

1. Initialize a trainable prefix P of length l and dimension d (matching the model's hidden dimension).
2. For each input x, concatenate P with the input embedding: $[P; \text{embed}(x)]$.
3. Process the concatenated sequence through the model as usual.
4. During training, optimize only the parameters of P, keeping the rest of the model frozen.

2.4.2 Selective Methods

Selective Methods focus on the strategic fine-tuning of a small, carefully chosen subset of the model's existing parameters. Instead of augmenting the model with additional components, this approach involves identifying and tuning only the most relevant parameters within the model's existing architecture. This often includes parameters within specific layers, such as biases or particular neurons, thereby minimizing the number of trainable parameters.

BitFit

BitFit, introduced in [58], is a notable example of a selective method where only the bias terms of the model are fine-tuned. All other parameters within the model are frozen during

the fine-tuning process. Despite the simplicity of this approach, BitFit has demonstrated competitive performance across various tasks, offering a cost-effective and resource-efficient alternative to more extensive fine-tuning methods.

The BitFit process can be described as follows:

1. Identify all bias terms in the model (typically found in linear layers, layer normalizations, and embedding layers).
2. During fine-tuning, update only these bias terms while keeping all other parameters (including weights) frozen.
3. The objective function remains the same as in full fine-tuning, but the gradient updates are applied only to the selected bias terms.

Mathematically, for a linear layer $y = Wx + b$, only b is updated during BitFit, while W remains constant.

2.4.3 Reparameterization Methods

Reparameterization Methods involve transforming the model's parameters into a lower-dimensional space, where fine-tuning is conducted. The underlying premise is that many parameters in large-scale models exhibit redundancy and can be effectively represented in a more compact form. This transformation allows for fine-tuning in a reduced parameter space, thus lowering the computational and memory demands of the process.

LoRA (Low-Rank Adaptation)

LoRA, or Low-Rank Adaptation, introduced in [15], exemplifies a reparameterization-based approach. This method decomposes the weight matrices of the model into lower-dimensional components, enabling fine-tuning to occur within this compact representation. LoRA is particularly beneficial for large-scale models, as it provides significant reductions in the number of trainable parameters while preserving, or even enhancing, the model's performance.

The key idea behind LoRA is to represent the weight update ΔW for a pre-trained weight matrix W as a low-rank decomposition:

$$\Delta W = BA \tag{2.9}$$

Where:

- B is a matrix of size $d \times r$
- A is a matrix of size $r \times k$
- r is the rank of the decomposition (typically much smaller than d and k)

During fine-tuning, instead of updating W directly, LoRA optimizes A and B. The final output is computed as:

$$h = x(W + BA) \tag{2.10}$$

Where x is the input to the layer.

2.5 Structured Knowledge Representation

The extraction of structured knowledge from unstructured resources is a fundamental challenge in the field of information management and artificial intelligence. This process relies heavily on understanding and implementing structured knowledge representation paradigms. These paradigms serve as the foundation for systematically organizing information in formats that are both human-readable and machine-processable, effectively bridging the gap between raw data and actionable insights.

Unlike unstructured data, such as free text, structured knowledge adheres to predefined schemas or models, which enable efficient storage, retrieval, and inference operations. By formalizing complex real-world concepts and their interrelationships, structured knowledge representation enhances data integration, interoperability, and the development of intelligent applications. This formalization is crucial for the development of data analysis tools, intelligent applications, and decision-making processes.

This section explores three fundamental approaches to structured knowledge representation: relational models, knowledge graphs and hypergraphs. Each paradigm offers unique strengths and addresses specific challenges in knowledge management and utilization.

2.5.1 Relational Models

Relational models have been a cornerstone of structured data representation and management since their introduction in [5]. These models have revolutionized the way structured knowledge is organized by arranging data into tables or relations. The relational model forms the basis of relational database management systems (RDBMS), which have become one of the most prevalent methods for storing and managing data across various industries.

At its core, the relational model represents data as a collection of tables, or "relations," where each table consists of rows (tuples) and columns (attributes). This seemingly simple structure belies a powerful mathematical foundation rooted in set theory and first-order predicate logic. By abstracting data into this tabular format, the relational model provides a uniform way to represent complex data structures and relationships.

One of the key strengths of the relational model lies in its ability to maintain data integrity and consistency. This is achieved through the use of keys, constraints, and the implementation of ACID properties (Atomicity, Consistency, Isolation, Durability). Primary keys uniquely identify each row within a table, while foreign keys establish relationships

between tables. These mechanisms, combined with other constraints like unique and check constraints, ensure data remains accurate and coherent across the entire database.

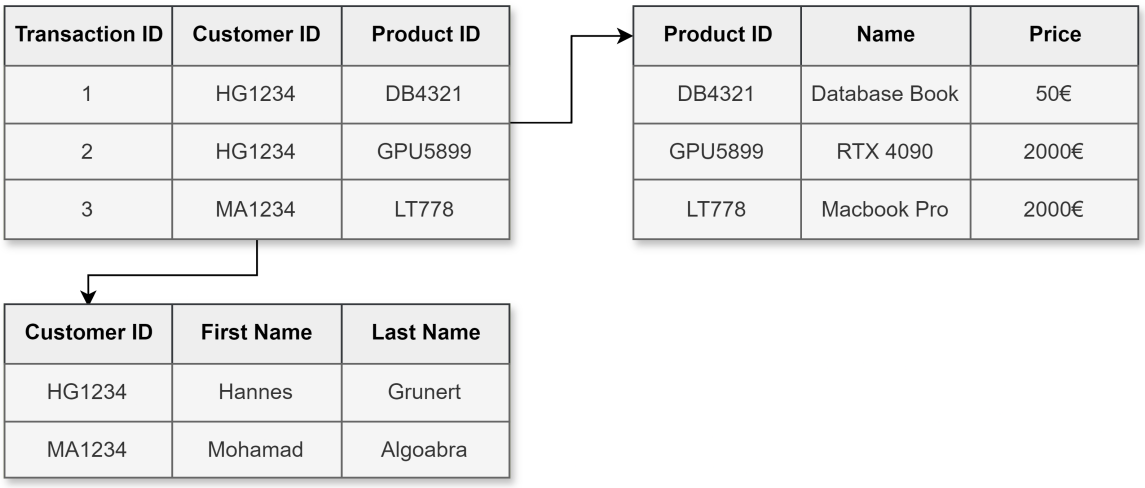


Figure 2.8: Simple relational database example

The relational model also introduces a standardized way to query and manipulate data through relational algebra and calculus. Relational algebra provides a set of operations such as selection, projection, and join, which can be combined to form complex queries. Relational calculus, on the other hand, offers a declarative approach to specifying queries based on mathematical logic. These theoretical underpinnings form the basis for SQL (Structured Query Language), the de facto standard for interacting with relational databases.

However, relational models are not without limitations, particularly when handling highly interconnected or rapidly changing data structures. Their rigid schema can become restrictive as data complexity increases. Furthermore, the need to manage semi-structured and unstructured data, along with demands for more intuitive representations of complex relationships, has driven the development of alternative approaches such as NoSQL databases[7] and knowledge graphs.

2.5.2 Knowledge Graphs

Knowledge graphs represent a more flexible approach to structured knowledge, emphasizing the connections between entities. The term "knowledge graph" gained prominence following Google's 2012 announcement of their Google Knowledge Graph, which marked a significant shift in how search engines understand and present information [41]. Google described their Knowledge Graph as a way to provide "things, not strings," moving beyond simple keyword matching to understanding the relationships between real-world entities.

[7] NoSQL (Not Only SQL) databases are non-relational database systems designed to handle large volumes of diverse, rapidly changing data types with flexible schemas, offering scalability and performance advantages for certain use cases.

Unlike relational databases, knowledge graphs use nodes, edges, and properties to model entities and their relationships. This graph-based structure allows for rich semantic representation and the accommodation of diverse data types and relationships. Knowledge graphs excel in scenarios where understanding the context and connections between data points is paramount, such as in natural language processing, recommendation systems, and semantic search.

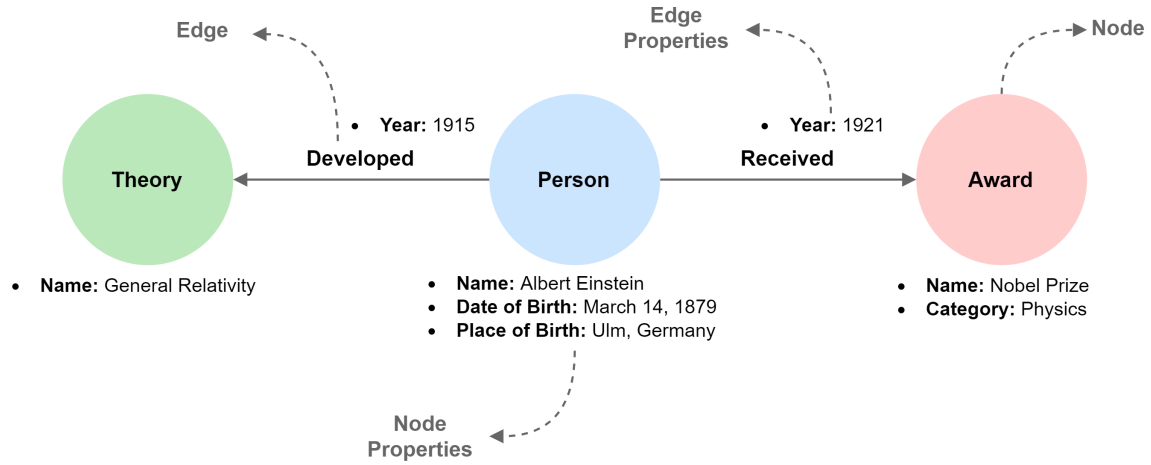


Figure 2.9: Simple knowledge graph example

The inherent flexibility of knowledge graphs enables the integration of heterogeneous data sources, supporting the creation of a unified view of information. They facilitate the discovery of new insights through graph-based algorithms, such as shortest path, centrality measures, and community detection, which can uncover hidden patterns and relationships within the data. Knowledge graphs also support inferencing through ontologies and reasoning engines, allowing for the derivation of implicit knowledge from explicitly stated facts.

Recent advancements in knowledge graph technologies have led to their adoption in various domains, including bioinformatics, social network analysis, and artificial intelligence. The development of standardized formats like RDF (Resource Description Framework) and graph databases like Neo4j have further enhanced the interoperability and accessibility of knowledge graphs.

2.5.3 Hypergraphs

Hypergraphs extend the concept of traditional graphs by allowing edges, known as hyperedges, to connect more than two nodes. This extension provides a more nuanced representation of complex relationships that cannot be adequately captured by simple pairwise connections in traditional graphs or knowledge graphs. In a hypergraph, each hyperedge can connect an arbitrary number of nodes, making it a powerful tool for modeling higher-order relationships.

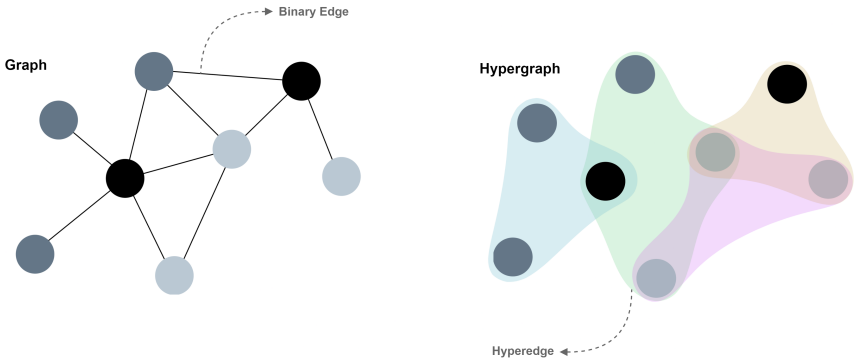


Figure 2.10: Simple hypergraph example

Hypergraphs are particularly useful in domains where relationships are inherently multi-faceted and cannot be easily decomposed into binary associations. For example, in a social network, a hypergraph could represent a group of individuals participating in a single conversation, rather than just pairwise friendships. Similarly, in biological networks, hypergraphs can model complex interactions involving multiple proteins, genes, or other biological entities simultaneously.

One of the key advantages of hypergraphs is their ability to naturally represent complex relationships without forcing them into simpler, potentially misleading forms. However, this complexity also introduces challenges, particularly in terms of computation and visualization. Despite these challenges, hypergraphs offer a rich framework for representing and analyzing data with intricate relationships.

The choice between relational models, knowledge graphs, and hypergraphs depends on the specific requirements of the knowledge extraction task, the nature of the unstructured resources, and the intended applications of the extracted knowledge. The table below summarizes the key differences between these paradigms.

Aspect	**Relational Databases**	**Knowledge Graphs**	**Hypergraphs**
Data Model	Tabular	Graph-based (nodes and edges)	Hypergraph-based (nodes and hyperedges)
Schema Flexibility	Rigid	Flexible	Flexible
Query Language	SQL	SPARQL, Cypher, Gremlin	Generalized hypergraph query languages (e.g., HyperGraphQL)
Scalability	High for structured data	High for interconnected data	High, but computationally intensive
Relation Type	Binary (two-way)	Binary (two-way)	Multi-way (n-ary)
Ideal Use Case	Structured, tabular data	Complex networks, semantic search	Complex, multi-entity relationships

Table 2.1: Comparison of relational databases, knowledge graphs, and hypergraphs

3 Literature Review

The field of knowledge retrieval and extraction from unstructured text stands at the intersection of two rapidly evolving areas of research: the development of increasingly powerful language models and the refinement of techniques for extracting structured information from text. This chapter provides a review of the current literature, highlighting key research contributions, identifying gaps in current knowledge, and positioning the present study within this dynamic landscape.

Our review is organized so that we first examine the models at our disposal - large language models (LLMs) themselves - before delving into the specific techniques and methods for structured knowledge extraction. This progression allows us to build a foundation for understanding the capabilities and limitations of current large language models, which will inform our subsequent exploration of how these models can be applied to the task of knowledge extraction.

3.1 Search Process

To ensure a detailed review of the literature, we employed a systematic search process designed to capture the most relevant and recent advancements in large language models and structured knowledge extraction techniques.

We began by selecting a diverse range of academic search engines to ensure broad coverage of both peer-reviewed and cutting-edge research. Our primary sources included Google Scholar for its comprehensive cross-disciplinary coverage, arXiv for the latest preprints in artificial intelligence and machine learning, ACL Anthology for specialized content in computational linguistics and natural language processing, IEEE Xplore for relevant research from engineering and computer science domains.

To guide our research, we began by establishing a list of primary keywords relevant to our objectives, such as "large language models", "LLM", "information extraction" and "knowledge extraction". To ensure we included the most current and promising models in our search, we looked at the LMSYS Arena Leaderboard[1]. This allowed us to identify top-performing models and subsequently search for any related research papers or documentation.

We then expanded our keyword list to include model-specific terms based on these findings, as well as task-specific terms like "named entity recognition", "relation extraction", and

[1]https://lmarena.ai/?leaderboard, accessed in 03. Sep. 2024

"knowledge graph generation". We used these terms in various combinations to conduct Boolean searches across the selected search engines. To ensure relevance and currency, we focused on publications from 2019-2024 and implemented a two-stage screening process. First, we conducted an initial title and abstract review to assess relevance to our research questions, followed by a full-text review of the papers that passed the initial screening.

3.2 Related Work

Our systematic search process revealed two primary areas of focus in the current literature: large language models and structured knowledge extraction techniques. By examining these areas considering their intersections, we aim to provide an overview of the current state of the field.

3.2.1 Foundational Language Models

Recent years have witnessed remarkable advancements in the development of LLMs, with several key players pushing the boundaries of natural language processing. This section focuses on the state-of-the-art models that have significantly impacted the field.

GPT Models

The GPT (Generative Pre-trained Transformer) family, developed by OpenAI, has consistently set new benchmarks in language modeling. GPT-3, introduced in [56], features an impressive 175 billion parameters and demonstrated unprecedented few-shot learning capabilities across various tasks. With 96 attention layers, 12,288 attention heads, and a 2048 token context window, GPT-3 was trained on 570GB of text data, setting a new standard for large-scale language models. Building on this success, GPT-4 was released in March 2023 [27], representing a significant leap in capabilities. While its exact architecture remains undisclosed, GPT-4 introduced multimodal capabilities, processing both text and images, and demonstrated enhanced performance in complex reasoning and creative tasks. It's estimated to have over 1 trillion parameters and boasts a 32,768-token context window. The latest iteration, GPT-4 omni, announced in May 2024, further improved performance and extended the context window to 128,000 tokens, while also offering a more cost-effective API for developers.

LLaMA Models

The LLaMA models, developed by Meta AI, represent a significant series of open-source large language models that have progressively advanced the field of natural language processing. The initial LLaMA [51], unveiled in February 2023, offered variants with 7, 13, 33 and 65 billion parameters. Trained on 1.4 trillion tokens and employing rotary positional embeddings (RoPE), these models showcased competitive performance with a 2048-token context window. Building upon this foundation, LLaMA 2 [50] was introduced in July 2023, with 7, 13 and 70 billion parameters. It extended the context length to 4096 tokens, leveraged 2 trillion tokens for training, and demonstrated enhanced instruction-following

capabilities. Then in April 2024 Meta have the released LLaMA 3 [9], featuring 8 billion and 70 billion parameter models trained on an impressive 15 trillion tokens. This iteration brought substantial architectural improvements, including grouped query attention (GQA) for faster inference and an attention mask to prevent cross-document self-attention. The most recent iteration, LLaMA 3.1, launched in July 2024, introduced a massive 405 billion parameter model alongside refined versions of the 8 billion and 70 billion parameter models. Trained on 15.6 trillion tokens, this update focused on enhancing multilingual proficiency, extending context handling to 128K tokens, and implementing more sophisticated safety measures [9].

Mistral Models

Mistral AI has made significant strides in the field of natural language processing through its open-source contributions, focusing on creating efficient and powerful language models. A key milestone in their progress was the release of Mistral 7B in September 2023 [18]. This model, with came with 7 billion parameters and incorporates a sliding window attention mechanism that optimizes the processing of longer sequences. Additionally, its 32,000-token context window allows for the management of extensive textual data, enabling it to outperform many larger models on various benchmarks. Building on this success, Mixtral 8x7B was introduced in December 2023, employing a Sparse Mixture-of-Experts (SMoE) approach [19]. This model boasts 46.7 billion total parameters but uses only 12.9 billion per inference, maintaining the 32,000-token context window while surpassing the performance of many 70 billion parameter models across a wide range of tasks. Mistral AI further advanced their technology with the release of Mistral Large Mixtral 8x22B [45], a SMoE model that uses only 39 billion active parameters out of 141 billion, offering unparalleled cost efficiency for its size. Recently, the company has expanded its portfolio with two new models: Mistral NeMo, a 12 billion parameter model developed in collaboration with NVIDIA [47], and Mistral Large 2, an impressive 123 billion parameter model [46].

Gemma Models

The Gemma models, developed by Google, marked their entry into the open-source large language model (LLM) ecosystem with their release in February 2024.The initial Gemma 1 models, available in 2 billion and 7 billion parameter versions, were trained on an extensive dataset of 6 trillion tokens. These models incorporate multi-query attention and GeGLU activations, offering an 8,192-token context window [44]. Building on this foundation, Google released Gemma 2 in mid-2024, featuring larger 9 billion and 27 billion parameter models. These new models also introduced enhanced safety features and tools like Shield-Gemma for content filtering and Gemma Scope for model interpretability, reinforcing Google's commitment to responsible AI development [43]. In addition to the open-source Gemma series, Google also developed its proprietary, closed-source Gemini models.

It's worth noting that there are other significant players in the LLM landscape, such as Anthropic's Claude models. These proprietary, closed-source models have demon-

strated impressive capabilities in natural language understanding, generation, and complex reasoning tasks, though specific details about their architecture are not publicly disclosed.

3.2.2 Structured Knowledge Extraction

The field of structured knowledge extraction has seen significant advancements in recent years, particularly with the application of large language models. This section explores key techniques and methodologies that have emerged to tackle various aspects of knowledge extraction.

REBEL: Relation Extraction By End-to-end Language generation

REBEL, introduced in [17], represents a novel approach to relation extraction that leverages using an autoregressive sequence-to-sequence (seq2seq) model based on BART [23]. Unlike traditional methods that treat relation extraction as a multi-step pipelines that can propagate errors. REBEL simplifies this by treating RE as a single, end-to-end generation task, where relational triplets (subject, relation, object) are directly generated from text. The model is pre-trained on a large, automatically generated dataset and fine-tuned on specific RE tasks, achieving state-of-the-art performance across various benchmarks.

UniversalNER: Targeted Distillation from Large Language Models for Open Named Entity Recognition

UniversalNER, proposed in [59], is a method for distilling the named entity recognition (NER) capabilities of large language models like ChatGPT into smaller, more efficient models specialized for NER tasks. This method employs targeted distillation with mission-focused instruction tuning, which allows the distilled models to maintain a high level of performance in recognizing a diverse array of entity types across multiple domains, all without the need for direct supervision. The targeted nature of this distillation ensures that the resulting models are not only more computationally efficient but also more effective in handling specific NER challenges compared to general-purpose distillation techniques. The paper also introduces a large NER benchmark, with 43 datasets spanning 9 domains, to comprehensively evaluate these models.

LLM-NERRE: Structured Information Extraction from Complex Scientific Text

LLM-NERRE, introduced in [17], represents a sequence-to-sequence approach for joint named entity recognition and relation extraction for complex hierarchical information in scientific text using LLMs like GPT-3. The approach involves training the model on 100-500 pairs of annotated text (prompts) and corresponding structured outputs (completions) to perform joint NER and RE tasks. The model can output information in various formats allowing for the extraction of intricate relationships between entities across sentences in scientific abstracts. The method demonstrates strong performance, particularly in materials science tasks like linking dopants to host materials, cataloging metal-organic frameworks (MOFs), and general information extraction, outperforming traditional NER and RE methods in both precision and F1 scores.

Grapher: Knowledge Graph Generation From Text

This paper introduces Grapher, a system designed for efficiently generating Knowledge Graphs (KGs) from textual data [25]. The authors propose a two-stage process where the first stage involves generating graph nodes using a pre-trained language model (T5) fine-tuned for entity extraction, and the second stage focuses on constructing the edges between these nodes to form the complete graph. The system is designed to be end-to-end trainable, which allows for optimized information transfer between the node and edge generation stages, enhancing the overall efficiency and accuracy of the KG construction. The authors also address challenges like edge imbalance, proposing solutions such as focal loss and sparsifying the adjacency matrix to improve training performance. Evaluations on datasets like WebNLG 2020, NYT, and TEKGEN demonstrate that Grapher matches or surpasses state-of-the-art performance, particularly excelling in scenarios where text-based nodes and GRU-based edge generation are employed. Despite its strengths, the paper notes limitations related to the computational complexity of handling large graphs and the focus on English-language data, suggesting these as areas for future improvement.

3.3 Findings and Gaps

Our review of the current literature on large language models (LLMs) and structured knowledge extraction techniques reveals several key findings and identifies important gaps in the existing research.

In terms of findings, we observe a rapid evolution in Large Language Models (LLMs) capabilities, with models like GPT-4, LLaMA 3.1, and Mistral Large are setting new benchmarks with increased parameter counts, extended context windows, and multimodal capabilities. Concurrently, a drive for efficiency has spawned smaller yet capable models such as Mistral 7B, LLaMA 3.1 8B and the Gemma series. This progression is reshaping knowledge extraction, moving from traditional pipelines to integrated, end-to-end solutions. Innovative techniques like REBEL and LLM-NERRE harness LLMs' generative prowess for tasks, including relation extraction and hierarchical information parsing from scientific texts. Enhanced context handling and multilingual proficiency are enabling more comprehensive, diverse document analysis. The open-sourcing of advanced models is democratizing access to cutting-edge NLP capabilities, potentially catalyzing breakthroughs in knowledge extraction across various fields.

However, our review also uncovers significant gaps in the current research landscape. Despite the multimodal capabilities of advanced models like GPT-4, multi-modal knowledge extraction remains underexplored. A notable disparity exists in multilingual knowledge extraction, particularly for low-resource languages, despite LLMs' improved language capabilities. Current approaches often overlook the crucial role of knowledge schemas in data validation and consistency, potentially compromising data quality and integration. Moreover, complex knowledge extraction tasks, such as hypergraph extraction with attributes—essential for representing intricate relationships in fields like biology or social network analysis—remain largely unaddressed.

To provide a clearer comparison of the discussed LLMs and knowledge extraction approaches, we present the following tables: Table 3.3 illustrates the significant differences in

Model	Parameters	Context Window	Dataset Size	Key Features	Open Source
GPT-4/4o	Undisclosed	128,000	Undisclosed	Multimodal, Multilingual, advanced reasoning	No
LLaMA 3.1	8B, 70B, 405B	128,000	15.6T tokens	Multilingual, GQA, enhanced safety and reasoning	Yes
Mistral Large	123B	32,000	Undisclosed	Multilingual, Reasoning, cost-efficient	No (open weights)
Gemma 2	2B, 9B, 27B	8,192	6T+ tokens	Safety features, interpretability tools	Yes
Gemini 1.5	Undisclosed	Up to 1M	Undisclosed	Multimodal, long-context understanding, efficient scaling	No
Claude 3.5	Undisclosed	200,000	Undisclosed	Multimodal, advanced reasoning, improved task performance	No

Table 3.1: Comparison of large language models

model capabilities among leading LLMs. GPT-4, LLaMA 3.1, and Claude 3.5 stand out with their extensive context windows, potentially allowing for more coherent long-form text generation and improved understanding of extended contexts. The open-source nature of LLaMA 3.1 and Gemma 2 is noteworthy for research transparency and adaptability.

Approach	Entity	Relation	Attributes	Ontology	Key Technique
REBEL	Yes	Yes	No	No	End-to-end seq2seq generation
UniversalNER	Yes	No	No	No	Targeted distillation from LLMs
LLM-NERRE	Yes	Yes	Yes	No	Joint NER and RE using LLMs
Grapher	Yes	Yes	No	Partial	Two-stage graph generation with T5 model

Table 3.2: Comparison of knowledge extraction approaches

Table 3.3 showcases the varying capabilities of different knowledge extraction approaches. REBEL and LLM-NERRE demonstrate more comprehensive extraction capabilities, handling both entities and relations. LLM-NERRE goes a step further by also extracting attributes, making it particularly suitable for complex information extraction tasks. Grapher stands out as the only approach in this comparison that generates ontologies, which could be particularly valuable for building structured knowledge bases. UniversalNER, while focused solely on named entity recognition, employs an innovative targeted distillation technique that could potentially be extended to other extraction tasks.

Building upon our review of the current state in language models and knowledge extraction techniques, several key opportunities emerge for advancing the field. Notably, existing approaches lack robust schema validation, struggle with complex multi-entity relationships, and have limited support for attribute-rich knowledge representation. In the following chapter, we present our methodology that addresses these limitations by combining the advanced capabilities of modern LLMs with a structured approach for ontology generation and knowledge extraction. Our framework specifically targets the extraction of information while maintaining data quality through schema validation and ontological consistency.

4 Methodology

This chapter presents our methodology for extracting structured knowledge from unstructured textual data using Large Language Models (LLMs) and custom hypergraph representations. Our approach addresses the limitations of existing methods by developing a universal knowledge extraction system that leverages the broad knowledge base of LLMs through transfer learning, accomplished by fine-tuning the model for specific tasks. By synergistically combining advanced natural language processing techniques with sophisticated knowledge representation structures, we create a robust, adaptable knowledge extraction pipeline capable of operating across diverse domains and languages.

The key components of our methodology include:

1. Conceptual framework integrating LLMs with hypergraph representations
2. Modular system architecture for efficient knowledge extraction and representation
3. Data preparation pipeline for generating high-quality synthetic training data
4. Model fine-tuning procedure using Low-Rank Adaptation (LoRA) and task-specific adapters
5. Evaluation methodology to assess the system's performance

In the following sections, we will detail each of these components, explaining their roles, implementation strategies, and the rationale behind their integration into our overall methodology.

4.1 Conceptual Framework

Our conceptual framework serves as the theoretical foundation for our approach to extracting structured knowledge from unstructured textual data. This framework is built upon the integration of two primary pillars:

1. Large Language Models (LLMs)
2. Hypergraphs for knowledge representation.

The synergistic integration of these components, facilitated by transfer learning and fine-tuning, allows us to address the multifaceted challenges inherent in knowledge extraction and representation.

4.1.1 Large Language Models

At the core of our framework lie LLMs, representing the pinnacle of modern Natural Language Processing technologies. These models, trained on extensive and diverse text corpora, demonstrate an unprecedented ability to comprehend, generate, and manipulate human-like text across a wide range of contexts and tasks.

Our research hypothesizes that the vast knowledge base acquired by LLMs during their pre-training phase confers several critical advantages that make them uniquely suited for knowledge extraction tasks:

1. **Adaptability to Various Domains:** LLMs possess a broad understanding of numerous subjects, allowing them to quickly adapt to specific domains with minimal fine-tuning. This adaptability stems from their exposure to diverse topics during pre-training, enabling efficient transfer learning to new, specialized areas.

2. **Deep Contextual Understanding:** These models excel at grasping subtle nuances, implicit relationships, and complex contextual cues within text. Their ability to capture long-range dependencies and understand context at multiple levels facilitates more accurate and nuanced knowledge extraction.

3. **Multilingual Capability:** Many advanced LLMs demonstrate proficiency in multiple languages, enabling cross-lingual knowledge extraction.

For a more detailed discussion of these hypotheses, please refer to Section 1.1.2

We propose that these characteristics can directly translate into enhanced performance in three crucial areas of information extraction:

- **Entity Recognition:** LLMs can identify and categorize named entities across diverse domains with high accuracy. Their broad knowledge base allows them to recognize entities in various contexts, including rare or domain-specific instances that might challenge traditional entity recognition systems.

- **Relation Extraction:** The deep contextual understanding of LLMs enables them to detect and classify relationships between entities, even when these relationships are complex, implicit, or require background knowledge.

- **Attribute Identification:** LLMs excel at recognizing relevant properties of entities and relations, even when these attributes are not explicitly stated. Their ability to infer information based on context and prior knowledge enhances the completeness of extracted information.

We argue that these capabilities make LLMs particularly well-suited for knowledge extraction tasks, potentially outperforming traditional methods that often require extensive domain-specific or language-specific training.

4.1.2 Knowledge Representation

To complement the advanced processing capabilities of LLMs, we employ a custom hypergraph structure for knowledge representation. This approach addresses the limitations of traditional graph structures, which are confined to binary relationships, by allowing the representation of higher-order relationships involving multiple entities simultaneously. This makes our system particularly well-suited for capturing the intricate complexity of real-world knowledge.

In our approach we explicitly differentiate between binary edges and higher-order edges (hyperedges). This distinction serves two crucial purposes:

1. It allows us to delineate different levels of context within the knowledge representation.
2. It enables the LLM to extraction of more detailed and nuanced information from the text.

By maintaining this separation, we can effectively capture both simple, direct relationships through binary edges and complex, multi-entity interactions through hyperedges. This dual approach provides a more comprehensive and flexible representation of knowledge, adapting to various levels of complexity in the information being modeled.

Formal Definition

To precisely define our knowledge representation system, we introduce the hypergraph H:

$$H = (N, E, HE) \tag{4.1}$$

Where:

- N is a finite set of nodes.
  - **Definition:** Nodes represent individual entities or concepts such as persons, places, objects, or abstract ideas, and serve as the basic units of the knowledge representation.
  - **Structure:** Each node $n \in N$ is defined as a tuple $(id_n, type_n, A_n)$, where:
    - id_n: A unique identifier for the node.
    - $type_n \in N_T$: The type of the node, from the set of entity types N_T.
    - A_n: A set of key-value pairs (k, v), where k is an attribute and v is its value.
  - **Example**: In the sentence "Alice, a software engineer, has been working at Microsoft since 2006.", two nodes would be identified:
    - $n_1 = (\{id_{\text{Alice}}, \text{Person}, \{\text{name} : \text{"Alice"}, \text{profession} : \text{"software engineer"}\})$
    - $n_2 = (\{id_{\text{Microsoft}}, \text{Company}, \{\text{name} : \text{"Microsoft"}\})$

- $E \subseteq N \times N$ is a set of binary edges.
  - **Definition:** Binary edges represent direct relationships between two entities.
  - **Structure:** Each edge $e \in E$ is defined as a tuple $(id_e, source, target, type_e, A_e)$, where:
    - id_e: A unique identifier for the edge.
    - $source, target \in N$: The source and target nodes in the relationship.
    - $type_e \in R_T$: The type of the edge, from the set of relation types R_T.
    - A_e: A set of key-value pairs (k, v), where k is an attribute and v is its value.
  - **Example**: Continuing with the previous sentence, a binary edge would be:
    - $e = (id_3, \{id_{\text{Alice}}, \{id_{\text{Microsoft}}, \text{"works\_for"}, \{\text{start\_year} : 2006\})$
- $HE \subseteq \mathcal{P}(N)$ with $|S| \geq 3$ for all $S \in HE$ is a set of hyperedges.
  - **Definition:** Hyperedges represent complex higher-order relationships involving multiple entities simultaneously.
  - **Structure:** Each hyperedge $he \in HE$ is defined as a tuple $(id_{he}, nodes, type_{he}, A_{he})$, where:
    - id_{he}: A unique identifier for the hyperedge.
    - $nodes \subseteq N$: The set of nodes connected by the hyperedge.
    - $type_{he} \in HE_T$: The type of the hyperedge, from the set of hyperedge types HE_T.
    - A_{he}: A set of key-value pairs (k, v), where k is an attribute and v is its value.
  - **Example**: For the sentence "Alice, Bob, and Charlie attended a conference in Berlin on October 5th.", a hyperedge would be:
    - $he = (id_4, \{id_{\text{Alice}}, id_{\text{Bob}}, id_{\text{Charlie}}, id_{\text{Berlin}}\},$ "conference_attendance", {date : "October 5th"})

This formal definition provides a rigorous mathematical foundation for our knowledge representation system that can capture both nuanced, multifaceted aspects of complex knowledge and straightforward connections ensuring clarity and consistency in its implementation.

4.1.3 Ontology and Validation

To enhance the robustness and reliability of our knowledge extraction process, we introduce an ontology generation and validation component. This step addresses potential inconsistencies or errors in the extracted knowledge that may occur due to the autoregressive nature of LLMs. The ontology defines the structure and constraints of our knowledge representation, guides the extraction process, validates the extracted knowledge, and ensures consistency and interoperability of the extracted information.

Formal Definition

To provide a foundation for our ontology, we define it formally as a tuple:

$$O = (N_T, R_T, HE_T) \tag{4.2}$$

Where:

- $N_T = n_1, n_2, \ldots, n_n$ is the set of entity types, where each n_i is a tuple (type, attributes).
- $R_T = r_1, r_2, \ldots, r_m$ is the set of binary relation types, where each r_i is a tuple (type, domain, range, cardinality, attributes).
- $HE_T = he_1, he_2, \ldots, he_k$ is the set of hyperedge types, where each he_i is a tuple (type, possibleEntities, attributes) and possibleEntities $\subseteq N_T$.
- Each a_i in $A = a_1, a_2, \ldots, a_j$ is a tuple (name, dataType, isOptional).

This formal definition provides a clear and structured representation of our ontology, laying the groundwork for the knowledge extraction and validation process.

Validation Process

The ontology $O = (N_T, R_T, HE_T)$ serves as a schema for validating the extracted knowledge representation $H = (N, E, HE)$. The validation process ensures the following conditions:

1. **Type Consistency:** Ensures that all elements in the knowledge representation conform to the types defined in the ontology.
   - For nodes: $\forall n \in N, \text{type}_n \in N_T$
   - For edges: $\forall e \in E, \text{type}_e \in R_T$
   - For hyperedges: $\forall he \in HE, \text{type}_{he} \in HE_T$

   Explanation: This check prevents the inclusion of undefined types, maintaining the structural integrity of the knowledge representation.

2. **Attribute Consistency:** Verifies that all attributes of nodes, edges, and hyperedges adhere to the definitions in the ontology.

$$\begin{aligned} &\forall x \in (N \cup E \cup HE),\ \forall (k, v) \in x.A, \\ &\exists a \in A :\ a.\text{name} = k \land x.\text{type} \in a.\text{appliesTo} \land v \text{ conforms to } a.\text{dataType} \end{aligned} \tag{4.3}$$

   Explanation: This ensures that elements only have attributes that are defined for their type, with values of the correct data type, preventing inconsistencies in the data structure.

3. **Structural Integrity:** Checks that the relationships in the knowledge representation maintain the structural rules defined in the ontology.
   - For edges: $\forall e \in E, \exists r \in R_T : r.\text{type} = \text{type}_e \wedge \text{type}_{\text{source}} \in r.\text{domain} \wedge \text{type}_{\text{target}} \in r.\text{range}$
   - For hyperedges: $\forall he \in HE, \forall n \in he.\text{nodes}, \text{type}_n \in he_i.\text{possibleEntities} : he_i.\text{type} = \text{type}_{he}$

   Explanation: This ensures that relationships connect appropriate types of entities, maintaining the semantic structure of the knowledge representation.

4. **Cardinality Constraints:** Enforces the maximum number of relationships an entity can participate in, as defined in the ontology.

$$\begin{aligned} \forall r \in R_T, \forall n \in N, |\{e \in E \mid e.\text{type} = r.\text{type} \wedge \\ (e.\text{source} = n \vee e.\text{target} = n)\}| \leq r.\text{cardinality} \end{aligned} \tag{4.4}$$

   Explanation: This prevents over-connection of entities, ensuring that the knowledge representation doesn't violate real-world constraints modeled in the ontology.

5. **Mandatory Attribute Presence:** Checks for the presence of all required attributes in entities, relations, and hyperedges.

$$\begin{aligned} \forall a \in A : a.\text{isOptional} = \text{false}, \forall x \in (N \cup E \cup HE) : \\ x.\text{type} \in a.\text{appliesTo}, \exists (k, v) \in x.A : k = a.\text{name} \end{aligned} \tag{4.5}$$

   Explanation: This ensures that all necessary information is present, maintaining the completeness of the knowledge representation.

6. **Uniqueness Constraints:** Prevents duplication of entities with unique identifiers in the knowledge representation.

$$\forall n_1, n_2 \in N, n_1.\text{id} = n_2.\text{id} \Rightarrow n_1 = n_2 \tag{4.6}$$

   Explanation: This maintains the integrity of the knowledge representation by ensuring each unique entity is represented only once.

This tight coupling between the knowledge representation and the ontology ensures that all extracted knowledge adheres to the predefined structure and constraints. By enforcing these rules, we maintain consistency and facilitate integration with existing knowledge bases. This validation step is crucial in preserving the integrity of our knowledge hypergraph, especially when handling potentially noisy or inconsistent output from LLMs.

4.1.4 Conceptual Workflow

Building upon the foundations laid out in our conceptual framework, we now present a detailed workflow that illustrates how our system operationalizes the integration of

LLMs with hypergraph representations for knowledge extraction. This workflow consists of several key steps that ensure a robust, accurate, and validated knowledge extraction process.

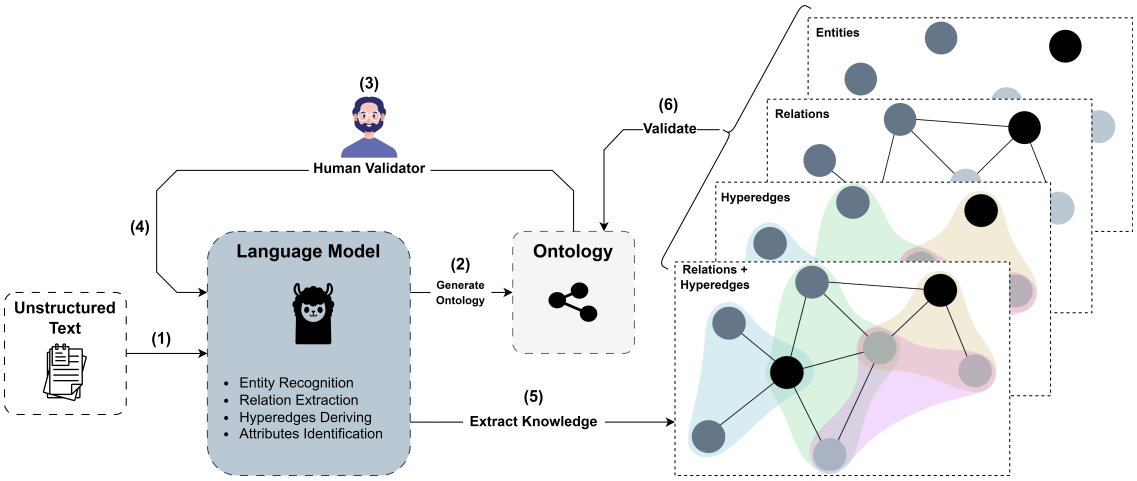


Figure 4.1: Conceptual workflow

As depicted in the Figure 4.1, the process involves multiple stages, from initial ontology generation through to the final production of a validated knowledge hypergraph. We will now detail each step of this workflow:

1. **Ontology Generation:** The workflow begins with the generation of an ontology from the given input text. This step leverages the LLM's natural language understanding capabilities to identify and structure the key concepts, relationships, and attributes present in the domain of interest. The ontology serves as a schema for the knowledge to be extracted, defining the types of entities, relations, and hyperedges that are relevant to the domain.

2. **Human Validation of Ontology:** Once the initial ontology is generated, it undergoes a crucial human validation step. This ensures that the ontology accurately reflects the domain knowledge and includes all the relevant information that needs to be extracted. Human experts can refine, add, or remove elements from the ontology to align it with the specific requirements of the knowledge extraction task.

3. **Knowledge Extraction:** With the validated ontology in place, the system proceeds to extract knowledge from the input text in an end-to-end manner. This process utilizes the LLM's advanced natural language processing capabilities, guided by the structure defined in the ontology. Unlike traditional pipeline approaches, our system performs a joint extraction of entities, relations, hyperedges, and their associated attributes. This integrated approach allows for:

    a) *Entity and Attribute Identification:* The system identifies entities mentioned in the text, classifying them according to the ontology-defined types. Simultaneously, it extracts relevant attributes for each entity, populating the attribute sets as defined in the ontology.

b) *Binary Relation Extraction:* As entities are identified, the system recognizes binary relationships between them. These relationships are classified according to the relation types specified in the ontology, and their attributes are extracted concurrently.

c) *Hyperedge Identification:* Complex, multi-entity relationships are identified and structured as hyperedges. The system determines which entities participate in each hyperedge and extracts the relevant attributes as defined in the ontology.

This joint extraction approach leverages the LLM's ability to understand complex textual relationships and the structured guidance provided by the ontology. It results in a more coherent and comprehensive knowledge extraction, capturing the intricate interplay between entities, relations, and higher-order relationships present in the text.

4. **Hypergraph Construction:** As knowledge is extracted, the system simultaneously constructs a custom hypergraph representation. This hypergraph, as formally defined in Section 4.1.2, consists of nodes (entities), binary edges (binary relations), and hyperedges (complex, multi-entity relations). Each element in the hypergraph is populated with the extracted information and attributes.

5. **Ontology-based Validation:** The constructed hypergraph undergoes a rigorous validation process against the ontology, as detailed in Section 4.1.3. This validation ensures:

   - Type consistency of all elements
   - Attribute consistency and completeness
   - Structural integrity of relationships
   - Adherence to cardinality constraints
   - Presence of mandatory attributes
   - Enforcement of uniqueness constraints

   This step is crucial for identifying and flagging any inconsistencies, errors, or missing information in the extracted knowledge.

6. **Refinement and Finalization:** Based on the results of the ontology-based validation, the system may iterate on the knowledge extraction process to address any identified issues. This could involve re-querying the LLM for specific pieces of information, adjusting extraction parameters, or flagging areas that require human intervention. Once all validation criteria are met, the final knowledge hypergraph is produced.

This workflow exemplifies the synergy between LLMs and hypergraph representations in our system. It leverages the strengths of LLMs in understanding and extracting complex information from natural language, while using the structured representation and validation capabilities of hypergraphs and ontologies to ensure the accuracy and consistency of the extracted knowledge.

4.2 System Architecture

The system architecture presented herein serves as the operational framework for implementing our conceptual approach, facilitating the integration of LLMs with hypergraph representations for knowledge extraction. This architecture translates theoretical concepts into a practical, implementable system, comprising four primary components that function in concert to process input data, fine-tune the model, extract knowledge, and integrate it into a database.

Figure 4.2 provides a schematic representation of this architecture, illustrating the interrelationships between components and the data flow through the system during both the training and inference phases.

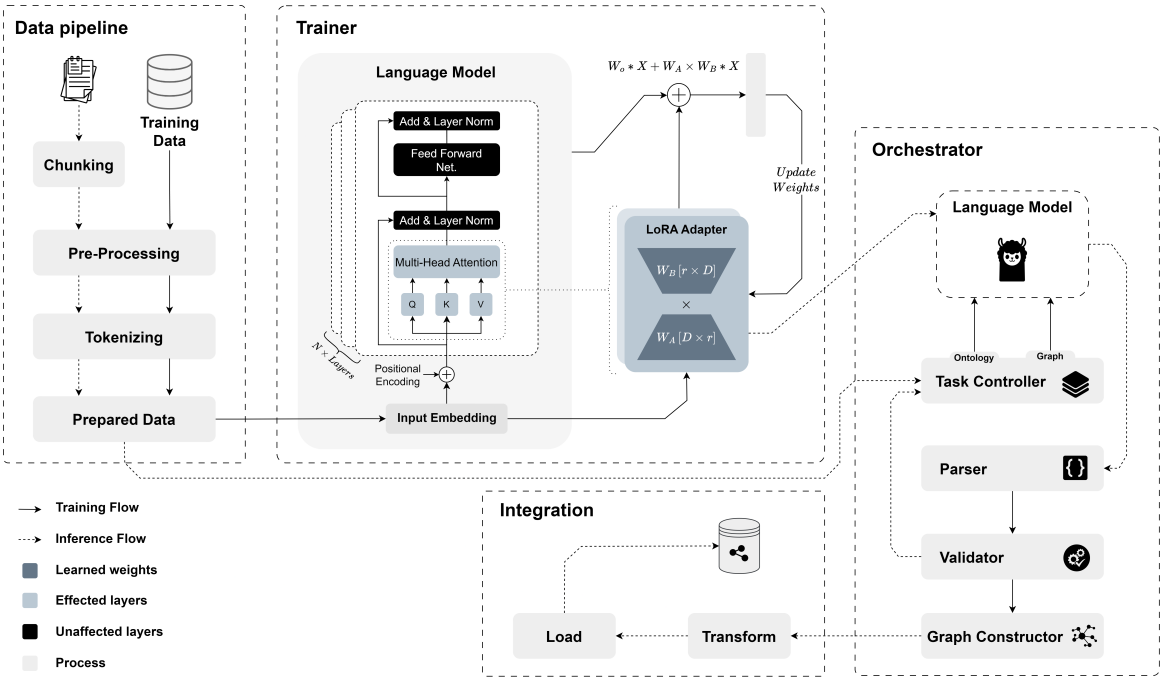


Figure 4.2: Architecture

The Data Pipeline functions as the initial stage of the system, responsible for data preparation and processing. This component ensures the optimal formatting of input data for compatibility with the LLM, encompassing tasks such as document chunking, text pre-processing, and tokenization. The Data Pipeline plays a crucial role in maintaining consistency between training and inference data formats, thereby establishing a foundation for effective knowledge extraction.

Central to the system is the Trainer component, which adapts the base LLM to the specific tasks of ontology generation and knowledge extraction. Utilizing Low-Rank Adaptation (LoRA) fine-tuning techniques, this component efficiently tailors the model's capabilities without necessitating adjustments to all parameters. The incorporation of task-specific

adapters further enhances the model's proficiency in the target tasks.

The LLM Orchestrator serves as the system's control center, managing the operational flow and ensuring the correct execution of tasks. This component coordinates the sequence of ontology generation and knowledge extraction, parses the LLM's outputs into structured formats, validates the extracted data against the predefined ontology, and constructs the final hypergraph representation. The LLM Orchestrator is instrumental in maintaining the integrity and consistency of the extracted knowledge.

The Knowledge Integration component facilitates the transition from extracted knowledge to practical application. It transforms the extracted data into a format compatible with the graph database, efficiently loads this information, and enables retrieval for various downstream tasks. This component ensures that the insights derived from unstructured text can be readily accessed and utilized in a structured form.

The subsequent subsections will provide a detailed examination of each architectural component, elucidating their functionalities, implementation strategies, and roles within the overall system.

4.2.1 Data Preparation

Data is the cornerstone of any machine learning system, and this is especially true for our knowledge extraction framework. The quality, diversity, and scale of the training data directly impact the performance and generalizability of the model. Knowledge extraction tasks present unique challenges, including the need for diverse domain coverage and the ability to handle complex, multi-entity relationships. Recognizing these challenges, we have developed a data preparation pipeline that emphasizes the creation of high-quality, diverse, and domain-spanning synthetic data.

Data Creation

To overcome the limitations of existing datasets and to ensure our model can handle a wide range of scenarios, we implemented a synthetic data generation approach. This method allows us to create a large, diverse, and controlled dataset that covers multiple domains and linguistic styles. Our process involves the following steps:

1. **Template Creation:** We leveraged state-of-the-art large language models, specifically GPT-4o[1] and Claude 3.5[2], to generate different templates across various domains. Each template consists of three components:
    - **Text Template:** An unstructured narrative or content with placeholders for variable information, serving as the base for generating diverse text samples.

[1] https://openai.com/index/hello-gpt-4o/, accessed in 20. Oct. 2024
[2] https://anthropic.com/news/claude-3-5-sonnet, accessed in 20. Oct. 2024

- **Ontology Template:** A structured representation of the knowledge schema, defining the categories, properties, and relationships to be extracted from the text, as outlined in Section 4.1.3.
- **Hypergraph Template:** A representation of the extracted knowledge in the form of a hypergraph, instantiated based on the information from both the ontology and text templates, as described in Section 4.1.2.

These templates are designed with variables that can be adjusted to create multiple variations, enhancing the diversity of our dataset.

2. **Manual Validation:** Despite the capabilities of large LLMs, we noticed that they can still make mistakes. To ensure the highest quality of our training data, we manually validated each template. This step was crucial in maintaining the accuracy and reliability of our synthetic data.

3. **Variation Generation and Data Augmentation:** From each validated template, we created multiple variations by employing several augmentation techniques:
   - **Linguistic Variations:** We introduced synonyms, paraphrases, and alternative sentence structures to help the model generalize across different writing styles.
   - **Variable Alteration:** We systematically modified the variables within each template to create diverse scenarios.
     - **Entity Substitution:** We replaced entities in our templates with others of the same type, creating new, valid scenarios.
     - **Relationship Permutations:** For templates with multiple entities, we generated permutations of the relationships between these entities.
     - **Context Expansion and Contraction:** We varied the amount of context provided in each example.
   - **Cross-lingual Augmentation:** To further diversify our dataset and improve the model's multilingual capabilities, we translated the templates into German and then back to English. We chose German due to its structural differences from English, which introduces valuable linguistic variations. This process helped in creating a more robust, language-agnostic model. In future iterations, we plan to expand this to include other languages, particularly those with non-Latin scripts.

   Each of these steps was done to enhance the model's ability to handle different variations in language and context. For instance, linguistic variations help the model become robust to different writing styles, while relationship permutations ensure the model can handle complex, interconnected knowledge structures. The cross-lingual augmentation specifically addresses the challenge of creating a model that can generalize across languages and capture language-independent knowledge structures.

4. **Negative Example Creation:** To enhance the model's ability to accurately extract knowledge and respect the given ontology, we generated negative examples. These

examples are designed to teach the model to handle cases where the text does not contain all the information specified in the ontology, or where it contains different information. This approach helps the model learn to:

- Identify when required information is missing from the text.
- Avoid extracting incorrect or unrelated information to fill ontology slots.
- Respect the structure and constraints of the given ontology.
- Handle partial matches and incomplete information gracefully.
- Distinguish between relevant and irrelevant information in the text.

For instance, we might provide a text about a person's career, paired with an ontology that asks for information about their education. The model should learn to leave the education fields empty rather than filling them with unrelated career information.

5. **Quality Assurance:** We implemented a rigorous quality assurance process to maintain the highest standards of data quality:
   - **Automated Checks:** We developed scripts to verify the consistency between the text, ontology, and hypergraph components of each data point.
   - **Manual Review:** We regularly reviewed samples from the generated dataset to ensure accuracy and relevance.
   - **Iterative Refinement:** Based on the findings from automated checks and expert reviews, we continuously refined our template generation and augmentation processes.

By employing this synthetic data generation approach, we created a rich, diverse, and high-quality dataset that forms the foundation of our knowledge extraction system. This data-centric approach not only enhances the performance of our model but also improves its generalizability across domains and languages, a crucial factor in real-world applications of knowledge extraction systems.

Data Pipeline

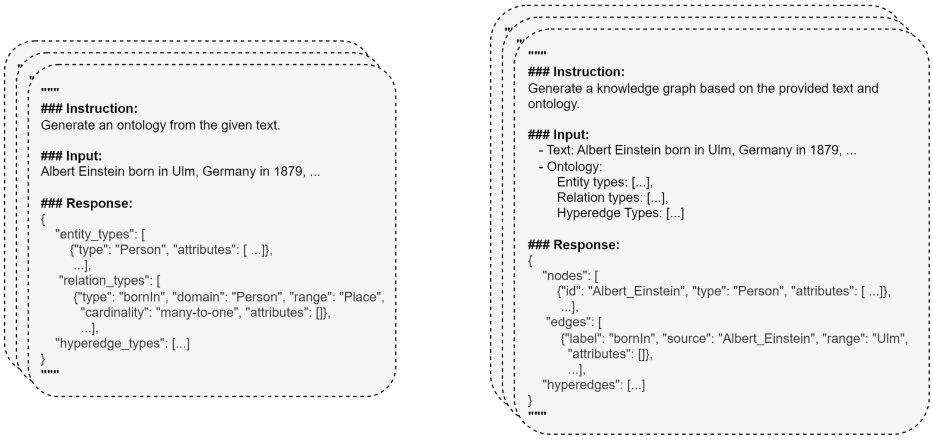
Following the creation of our synthetic dataset, we implemented a preprocessing pipeline to transform the raw data into a format optimized for model training and inference. This crucial stage ensures data consistency and quality, enabling our Large Language Model (LLM) to focus on learning the intricacies of knowledge extraction. Our preprocessing pipeline comprises the following key stages:

1. **Data Ingestion and Structuring:** We begin by ingesting the synthetic data from JSON files, carefully extracting and structuring the core components: unstructured text, ontology, and knowledge hypergraph. For each data point, we create two distinct training examples:
   - An ontology generation task, where the model learns to derive a structured ontology from raw text.

- A knowledge hypergraph extraction task, where the model learns to construct a knowledge hypergraph based on both the text and its corresponding ontology.

This dual-task approach allows our model to learn these interconnected skills independently while maintaining their inherent relationships.

2. **Prompt Formatting:** Next, we format our data into a specific prompt style that is appropriate for training the LLM. We adopt the Alpaca prompt format, which provides a clear structure for the instruction, input, and expected output providing a clear and consistent framework for each task. This format delineates three key sections:
   - Instruction: A clear, task-specific directive for the model.
   - Input: The context or information on which the model should base its response.
   - Response: The target output the model should generate.



(a) Ontology generation training instance

(b) Graph generation training instance

Figure 4.3: Training instances

For ontology tasks, the instruction guides the model to generate an ontology, with the input being the raw text. For knowledge graph tasks, the instruction prompts the creation of a knowledge graph, with both the original text and its ontology as input. By using a consistent prompt structure, we help the model understand the task requirements and expected output format more effectively.

3. **Tokenization and Encoding:** After formatting the data, we tokenize it using the same tokenizer used for the base model to maintain semantic consistency between pre-training and fine-tuning phases. This step is crucial because using a different tokenization method could lead to the model struggling to understand token meanings, resulting in poor performance. The tokenization process includes:
   - Converting text into token IDs recognizable by the model.
   - Handling special tokens (e.g., start-of-sequence, end-of-sequence) to delineate different sections of the input.
   - Dynamic padding and truncation to manage varying input lengths.

4. **Dataset Finalization:** In this final stage, we transform our processed data into a format directly consumable by our model training pipeline. This involves:
    - Converting tokenized data into tensor format.
    - Generating attention masks to differentiate between actual input and padding.
    - Organizing the data into batches optimized for the training.

4.2.2 Model Fine-Tuning

The adaptation of the LLM to the specific demands of knowledge extraction represents a critical step in our methodology. This process of fine-tuning not only tailors the model's vast capabilities to our particular use case but also serves as the bridge between the raw power of pre-trained language models and the nuanced requirements of structured knowledge extraction. In this section, we delineate our approach to model selection, elucidate our fine-tuning procedures, and detail our hyperparameter optimization strategy.

Base Model Selection

Our selection of the base model for our knowledge extraction system was predicated on an extensive literature review of recent advancements in LLMs, as detailed in section 3.3. This review covered a diverse range of models, including GPT, Mistral, Gemma, and the LLaMA family. Our decision focused on several key factors essential for our tasks:

- Model architecture and size
- Pre-training data and Performance
- Multilingual capabilities
- Computational requirements
- Licensing and accessibility for research

Based on our review, we selected LLaMA 3.1 8B as the foundation for our knowledge extraction system. This decision was driven by a combination of factors that position LLaMA 3.1 8B to meet the demands of our research objectives:

1. **Performance:** LLaMA 3.1 8B demonstrates outstanding performance across various benchmarks and tasks [9]. This performance is underpinned by its extensive pre-training on a massive dataset of 15.6 trillion tokens, giving the model a broad and deep knowledge base, which enhances its ability to understand and extract nuanced relationships from text across diverse domains.
2. **Size balance:** The 8B parameter version offers a good balance between computational efficiency and model capacity. This balance allows us to conduct experiments and use the model with reasonable resource requirements while still harnessing the benefits of increased capacity compared to smaller models.

3. **Multilingual capabilities:** LLaMA 3.1's significant advancements in cross-lingual transfer and multilingual understanding align perfectly with our ambition to create a knowledge extraction system capable of operating across linguistic boundaries.

4. **Expanded Context Window:** The model's ability to process longer sequences is critical for handling complex documents and extracting comprehensive knowledge graphs.

5. **Open-Source Nature:** The open-source availability of LLaMA 3.1 provides unparalleled flexibility in adaptation and fine-tuning, aligning with our immediate research goals and leaving ample room for future enhancements.

Fine-Tuning Procedure

To adapt LLaMA 3.1 8B for our knowledge extraction system, we leveraged a combined instruction fine-tuning with Low-Rank Adaptation (LoRA) to enhance the model's understanding of our knowledge representations and task specifications. We chose LoRA for its efficiency in parameter updates and its ability to preserve the model's pre-trained knowledge while minimizing the risk of catastrophic forgetting during adaptation to new tasks [15]. Our fine-tuning approach focused on two key tasks in our knowledge extraction pipeline:

1. Ontology generation from unstructured text

2. Knowledge hypergraph extraction based on text and ontology

For each of these tasks, we employ a unique combination of LoRA adapters and task-specific instructions. This integrated approach allows for task-specific optimization and switch between tasks efficiently while framing the problems as natural language instructions, making the model more flexible and capable of handling our tasks by changing the active adapter.

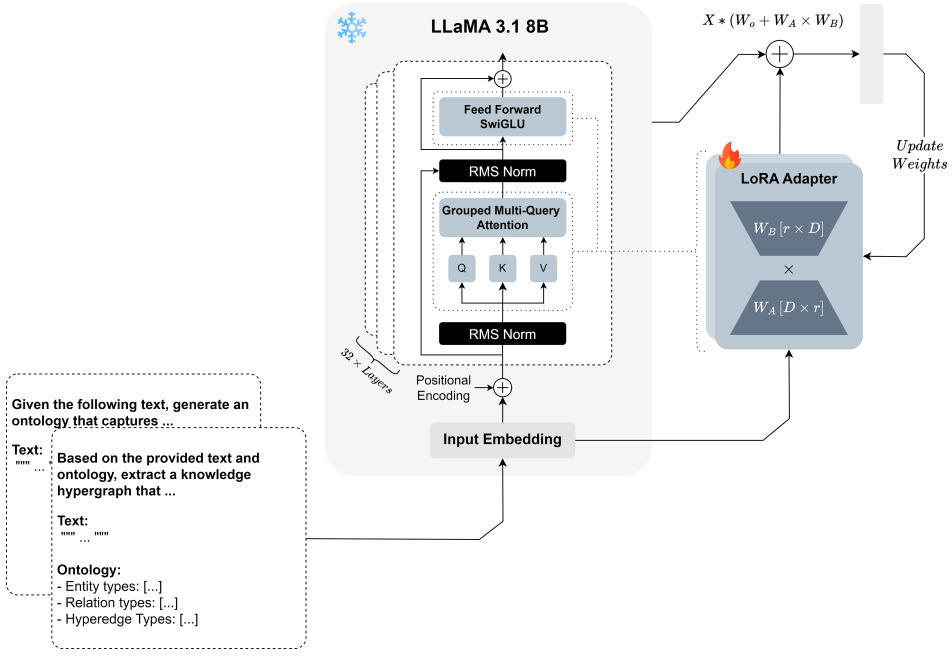


Figure 4.4: LLaMA 3.1 8B fine-tuning with LoRA

As illustrated in Figure 4.4, our fine-tuning process integrates LoRA's additional low-rank matrices, which act as compact, learnable transformations modifying key model components, with instruction-based task framing. We embed task-specific natural language prompts in the input sequence, as follows:

- Ontology generation: "Given the following text, generate an ontology that captures the key entities, their relations, higher-order relations, and associated attributes"
- Knowledge hypergraph extraction: "Based on the provided text and ontology, extract a knowledge hypergraph that represents the information and connections described."

This approach allows the LoRA adapters to learn task-specific modulations of the model's behavior, effectively specializing it for each task while maintaining the flexibility to switch between them by changing the active adapter and instruction. We can formally represent this process as an optimization problem as follows:

$$\min_{\theta_{LoRA}} \mathcal{L}(\theta_{LoRA}) = -\frac{1}{N} \sum_{i=1}^{N} \log p_{\theta_{LoRA}}(y_i|x_i, I_i) \tag{4.7}$$

Where:

- θ_{LoRA} represents the LoRA parameters (matrices W_A and W_B)
- N is the number of training examples
- x_i is the input text
- I_i is the task-specific instruction
- y_i is the expected output (e.g., ontology or knowledge hypergraph)
- $p_{\theta_{LoRA}}(y_i|x_i, I_i)$ is the probability of generating the correct output given the input and instruction

In our implementation, we focused on adapting the following layers:

- Query (Q), Key (K), Value (V) and the Output projections within the Grouped Multi-Query Attention[3] blocks
- Up and down projections in the Feed-Forward SwiGLU[4] blocks

By targeting these specific components, we achieve a balance between adaptation power and computational efficiency. The attention mechanism adaptations allow the model to better focus on relevant information for our knowledge extraction tasks, while the feed-forward adaptations enhance the model's capacity to perform complex reasoning on the attended information.

[3]Grouped Multi-Query Attention is an optimized variant of multi-head attention where queries are processed in groups, reducing computation and memory usage while maintaining model quality.

[4]Feed-Forward SwiGLU uses the SwiGLU (Swish-Gated Linear Unit) activation function in Transformer models' feed-forward layers, improving performance and representation over ReLU or GELU.

For each task in our knowledge extraction pipeline, we train a LoRA adapter, which consists of two matrices: $W_A[D \times r]$ and $W_B[r \times D]$, where:

- D is the dimension of the original weight matrix (4096 for LLaMA 3.1 8B)
- r is a hyperparameter determining the rank of the decomposition (typically much smaller than D)

These matrices are used to compute a low-rank update to the model's weights, modifying the fine-tuned model weight matrix as follows:

$$W_{ft} = W_o + \alpha * W_A \times W_B \tag{4.8}$$

Where:

- W_{ft} is the merged weight matrix used for inference after fine-tuning
- W_o represents the original pre-trained weights (frozen during fine-tuning)
- $W_A \times W_B$ forms a low-rank update that captures task-specific information with a reduced number of parameters.
- α is a scaling factor controlling the magnitude of the adaptation

This formulation allows us to update the model's behavior with only $2 * D * r$ trainable parameters per layer, significantly less than the full parameter count of the original model.

During training only W_A and W_B are updated, while W_o remains frozen. After training, the LoRA update can be merged with the original weights for efficient inference, or kept separate for easy task switching.

4.2.3 Knowledge Extraction

The Knowledge Extraction component, anchored by the LLM Orchestrator, forms the core of our system's ability to transform unstructured text into structured knowledge using our fine-tuned LLM. This component orchestrates a multi-step process that leverages the capabilities of our adapted LLaMA 3.1 8B model to generate ontologies and extract knowledge hypergraphs. The process is designed to be both flexible and capable of handling diverse text inputs while ensuring the extracted knowledge adheres to predefined structural and semantic constraints.

LLM Orchestrator

The LLM Orchestrator serves as the central control mechanism for the knowledge extraction pipeline. Its primary functions include:

1. **Task Coordination:** It manages the sequence of operations, beginning with ontology generation and progressing to knowledge hypergraph extraction. This ensures a smooth flow of data between different stages of the extraction process.

2. **Model Interaction:** It handles all communication with the fine-tuned LLM, including input preparation and switching between task-specific LoRA adapters for ontology generation and knowledge extraction.
3. **Output Parsing:** It parses the LLM's output, converting natural language responses into structured json format. This involves constructing ontology and knowledge hypergraph representations based on the model's output.
4. **Validation:** The Orchestrator ensures that the extracted knowledge adheres to the ontology as described in Section 4.1.3.
5. **Hypergraph Construction:** Finally, it assembles the extracted knowledge into a coherent hypergraph structure as defined in 4.1.2.

The knowledge extraction process, as managed by the LLM Orchestrator, follows these steps:

1. **Ontology Generation:**
   - The process begins with the generation of an ontology from the input text. This step utilizes the ontology generation LoRA adapter and corresponding instruction prompt.
   - The LLM Orchestrator constructs a prompt that includes the task-specific instruction: "Given the following text, generate an ontology that captures the key entities, their relations, higher-order relations, and associated attributes."
   - The fine-tuned model processes this prompt and generates a structured ontology, defining entity types, relation types, hyperedge types, and attributes as detailed in Section 4.1.3.
   - The Orchestrator then parses and validates the generated ontology, ensuring it adheres to the formal definition $O = (N_T, R_T, HE_T)$ as defined in Section 4.1.3.
   - The user then validates the ontology to ensure it meets all requirements.
2. **Knowledge Hypergraph Extraction:**
   - With the validated ontology in hand, the Orchestrator proceeds to the knowledge extraction phase, activating the knowledge hypergraph extraction LoRA adapter.
   - A new prompt is constructed, incorporating both the original text and the generated ontology, with the instruction: "Based on the provided text and ontology, extract a knowledge hypergraph that represents the information and connections described."
   - The model processes this input, leveraging its fine-tuned capabilities to identify entities, relations, and complex multi-entity relationships (hyperedges) as defined by the ontology.
   - The LLM Orchestrator parses the model's output, structuring it into the formal hypergraph representation $H = (N, E, HE)$ as defined in Section 4.1.2.

3. **Validation and Refinement:**
    - The extracted knowledge hypergraph undergoes a rigorous validation process against the ontology, as detailed in Section 4.1.3.
    - This validation checks for type consistency, attribute consistency, structural integrity, cardinality constraints, mandatory attribute presence, and uniqueness constraints.
    - If any inconsistencies or errors are detected, the Orchestrator initiates a refinement process. This may involve:
        - Re-querying the LLM with more specific prompts to address identified issues.
        - Flagging issues for human review, if necessary.
    - The refinement process may iterate multiple times until all validation criteria are met.
4. **Final Hypergraph Construction:**
    - Once all validation checks are passed, the LLM Orchestrator finalizes the construction of the knowledge hypergraph.
    - This involves assembling all validated nodes, edges, and hyperedges into a coherent structure, ensuring all relationships and attributes are correctly represented.
    - The final hypergraph is then prepared for integration into the knowledge database or for use in downstream applications.

4.2.4 Knowledge Integration

The Knowledge Integration component is responsible for incorporating the extracted knowledge into a graph database, making it accessible for various downstream applications. This process involves transforming the knowledge hypergraph into a format suitable for database storage, loading the data efficiently, and providing mechanisms for retrieval and querying.

Graph Database Selection

For our knowledge integration, we chose Neo4j as our graph database system. This decision was driven by Neo4j's native graph storage, which aligns well with our knowledge hypergraph structure and its flexible schema that adapts to evolving ontologies. Neo4j's query language, Cypher, offers powerful capabilities for traversing complex relationship patterns. However, Neo4j doesn't natively support hyperedges, therefore we've developed a strategy to implement them using a combination of nodes and relationships, which we'll detail in the data modeling section. These features make Neo4j an ideal choice for storing, querying, and analyzing our complex knowledge structures.

Data Modeling

To represent our knowledge hypergraph in Neo4j, we employ the following modeling approach:

- **Entities as Nodes:** Each entity extracted from the text is represented as a node in the graph. The node's properties store the entity's attributes.
- **Binary Relations as Relationships:** Direct relationships between two entities are modeled as Neo4j relationships, with relationship types corresponding to the relation types defined in our ontology.
- **Hyperedges as Hypernode:** To represent hyperedges (complex, multi-entity relationships), we use a hypernode pattern: Create a node to represent the hyperedge itself. Connect this node to all participating entities with relationships. Store hyperedge attributes as properties of the hypernode.
- **Ontology Integration:** We store the ontology as a separate subgraph within Neo4j, allowing for runtime validation and schema enforcement.

Data Transformation

Before loading into the graph database, our knowledge hypergraph undergoes a transformation process:

1. **Node Preparation:**
    - Generate unique identifiers for each entity.
    - Transform entity attributes into a property format compatible with Neo4j.
2. **Relationship Mapping:**
    - Convert binary edges in the hypergraph to Neo4j relationship structures.
    - Generate unique identifiers for relationships if required.
3. **Hyperedge Transformation:**
    - Create hypernode structures for each hyperedge.
    - Generate relationships between hypernodes and participating entities.

Data Loading

After transforming the knowledge hypergraph into a suitable format, we load the data into graph database. The loading process is divided into three main steps: creating nodes, creating edges, and creating hyperedges.

1. **Creating Nodes:** We start by creating or updating nodes in the Neo4j database. For each entity in our knowledge graph, we use a MERGE operation to ensure idempotency. This means that if a node with the given ID already exists, it's updated; otherwise, a new node is created. We set the node's type using a label and add its attributes as properties. This approach allows us to efficiently handle both new data and updates to existing entities.

2. **Creating Edges:** Next, we establish relationships (edges) between the nodes. For each edge in our data, we first match the source and target nodes using their unique IDs. Then, we create a relationship of the specified type between these nodes. Any additional attributes of the edge are set as properties on the relationship.

3. **Creating Hyperedges:** To represent hyperedges, we implement the hypernode pattern. For each hyperedge, we create a new node of type 'Hyperedge'. The hyperedge's attributes are set as properties on this node. We then create 'PART_OF_HYPEREDGE' relationships between this hyperedge node and all participating entity nodes.

The entire loading process is executed within the database transactions to ensure data consistency. We first create or update all nodes, then establish the binary edges between nodes, and finally create the hyperedge structures. This order of operations ensures that all necessary nodes exist before we attempt to create relationships between them.

4.3 Evaluation Techniques

The evaluation is essential to determine the effectiveness and reliability of our knowledge extraction system. This section outlines our approach to testing and measuring the performance of our methodology. We start by introducing our testing dataset, then we explain our evaluation metrics, which provide a nuanced understanding of the system's capabilities in terms of structural consistency, knowledge accuracy, and ontology adherence. Lastly, we discuss the specifics of our training process, including hyperparameters and architectural choices, to provide a complete picture of how these details can affect the performance of the system.

4.3.1 Testing Dataset

To evaluate our knowledge extraction system, we constructed a testing dataset of 370 samples in different domains using the synthetic data pipeline detailed in Section 4.2.1. The dataset is evenly divided, with 185 samples designated for evaluating ontology generation and 185 samples for assessing hypergraph extraction. This deliberate divergence in the domains ensures that our evaluation accurately reflects the system's ability to generalize and perform under varied conditions. Our testing dataset comprises synthetically generated data spanning multiple domains, each with distinct syntactic and semantic characteristics.

The domains include:

- **Healthcare**: Focuses on relationships between patients, healthcare providers, medical conditions, treatments, medications, and clinical outcomes. This domain captures interactions in medical records, treatment plans, and patient histories.
- **Legal**: Encompasses relationships between cases, legal documents, parties involved, jurisdictions, court decisions, and legal precedents. It represents the intricate network of legal proceedings and documentation.
- **Finance**: Covers scenarios between financial institutions, transactions, accounts, investments, market indicators, and regulatory compliance.
- **Recruitment**: Addresses scenarios between job seekers, employers, positions, skills, qualifications, and hiring processes. It captures the dynamics of workforce management and talent acquisition.
- **Education**: Represents relationships between students, instructors, courses, academic institutions, assessments, and learning outcomes. This domain models educational progression and academic performance tracking.
- **Other**: Encompasses miscellaneous domains with their unique entity relations, ensuring coverage of diverse knowledge extraction scenarios.

By encompassing multiple domains, the data ensures that the system is generalized and can effectively extract knowledge across different subject matters. This diversity is crucial for validating the system's universal applicability and transfer learning efficacy.

Each testing example is meticulously annotated with ground truth ontologies and knowledge hypergraphs. These annotations serve as benchmarks against which the system's outputs are compared, facilitating precise and objective evaluation of performance metrics.

4.3.2 Evaluation Metrics

To assess the performance of our knowledge extraction system, we employ a multi-faceted evaluation framework comprising three primary metrics: structure consistency, knowledge similarity, and ontology adherence. Each metric targets specific aspects of the system's output, ensuring a holistic evaluation.

1. **Structure Consistency:**

   Structure consistency evaluates the system's proficiency in generating ontologies and extracting knowledge in the correct format, specifically ensuring the validity of the JSON structure. This metric verifies that the foundational representation of knowledge adheres to the expected schema, which is critical for interoperability and seamless downstream processing.

This assessment involves two main checks:

- **Syntax Verification:** Checking for well-formedness and absence of structural errors in the JSON output.
- **Schema Validation:** Ensuring that the generated JSON complies with the predefined schema.

Formally, Structure Consistency is defined as the accuracy of outputs that comply with the expected schema as follows:

$$\text{SC} = \left(\frac{\sum_{i=1}^{N} S_i}{N} \right) \times 100\% \tag{4.9}$$

Where:

- SC is the Structure Consistency percentage.
- N is the total number of outputs.
- S_i is a binary variable indicating whether the i-th output conforms to the schema ($S_i = 1$) or not ($S_i = 0$).

2. **Knowledge Similarity:**

   Knowledge similarity measures the fidelity of the extracted knowledge by comparing it against ground truth annotations. This metric is bifurcated into three distinct evaluations, each capturing different dimensions of similarity between the extracted and reference knowledge bases:

   - **Exact Match:** This sub-metric quantifies the precision of entities, relations, hyperedges, and attributes extraction by calculating the proportion of correctly identified elements that exactly match the ground truth. An exact match implies that the extracted element precisely aligns with the reference in terms of identifier, type, and attributes without any deviations.

     Mathematically, Exact Match (EM) can be represented as:

     $$\text{EM} = \left(\frac{\sum_{i=1}^{M} E_i}{M} \right) \times 100\% \tag{4.10}$$

     Where:

     - EM is the Exact Match percentage.
     - M is the total number of elements in the ground truth.
     - E_i is a binary variable indicating whether the i-th element is an exact match ($E_i = 1$) or not ($E_i = 0$).
   - **Matcher Evaluation:** Assesses the system's ability to correctly identify and align elements that may not exactly match the ground truth but are contextually similar. This evaluation leverages string matching technique to account

for variations or minor discrepancies, thereby providing a more flexible and realistic measure of similarity.

The Matcher Evaluation uses the `SequenceMatcher` algorithm [35, 48], which computes a similarity ratio based on the longest contiguous matching subsequence (LCS) between two strings. The similarity score ranges between 0 and 1, where 1 indicates an exact match and 0 indicates no similarity.

Formally, the similarity score for the i-th element pair is defined as:

$$M_i = \frac{2 \times |LCS(\text{str}_i, \text{gt}_i)|}{|\text{str}_i| + |\text{gt}_i|} \tag{4.11}$$

Where:

- M_i is the similarity score for the i-th element.
- $LCS(\text{str}_i, \text{gt}_i)$ represents the Longest Common Subsequence between the extracted string str_i and the ground truth string gt_i.
- $|\text{str}_i|$ and $|\text{gt}_i|$ are the lengths of the extracted and ground truth strings, respectively.

- **Semantic Similarity:** Beyond exact and partial string matches, Semantic Similarity Assessment evaluates the contextual and conceptual alignment between the extracted knowledge and the ground truth. Leveraging embedding models, this sub-metric captures the semantic nuances and relationships that exact or partial matches might miss.

  By representing elements as high-dimensional vectors, the semantic similarity measures the cosine similarity between the embeddings of extracted elements and their corresponding ground truth elements. This approach ensures that even if the wording differs, semantically equivalent or related concepts are recognized as similar.

  Formally, for the i-th element, the semantic similarity score is defined as:

$$SS_i = \frac{v_i \cdot g_i}{\|v_i\| \times \|g_i\|} \tag{4.12}$$

  Where:

  - SS_i is the semantic similarity score for the i-th element.
  - v_i is the embedding vector of the extracted element.
  - g_i is the embedding vector of the ground truth element.
  - $\cdot$ denotes the dot product between two vectors.
  - $\|v_i\|$ and $\|g_i\|$ are the Euclidean norms of vectors v_i and g_i, respectively.

Overall, Knowledge Similarity provides a nuanced evaluation by combining exact matches, partial string similarities, and semantic alignments to comprehensively assess the accuracy and depth of the extracted knowledge.

3. **Ontology Adherence:** Ontology adherence assesses the extent to which the system respects the predefined ontology during the knowledge extraction process. This metric ensures that the extracted knowledge conforms to the structural and semantic constraints imposed by the ontology. The adherence score is computed based on the proportion of successfully validated elements relative to the total number of elements. For detailed information on the validation process, refer to section 4.1.3.

By employing these metrics, we ensure a comprehensive evaluation of our knowledge extraction system, capturing both quantitative and qualitative aspects of its performance. This multifaceted assessment enables the identification of strengths and areas for improvement, guiding future enhancements to the system.

4.3.3 Training Details

The training phase is critical in fine-tuning the base LLM to perform specialized tasks of ontology generation and knowledge hypergraph extraction. This section elucidates the specifics of our fine-tuning approach, including the selection of hyperparameters, the role of Low-Rank Adaptation (LoRA), and the impact of targeting different layers within the model.

We employed the pre-trained LLaMA 3.1 8B as our foundation, augmenting it with LoRA to enable efficient fine-tuning. LoRA introduces trainable rank decomposition matrices to key layers of the model, allowing for task-specific adaptations while maintaining most of the original pre-trained weights frozen as described in section 4.2.2.

Layer Targeting Strategy

We experimented with two primary approaches for applying LoRA:

1. **Attention-Only Adaptation:** In this configuration, we applied LoRA exclusively to the self-attention layers of the transformer blocks. This focused approach aims to enhance the model's ability to attend to task-relevant information without modifying the feed-forward processing stages.

2. **Comprehensive Adaptation:** This strategy extends LoRA application to both the self-attention and feed-forward layers, allowing for more extensive modifications to the model's processing pipeline. This approach potentially enables more nuanced adaptations at the cost of increased computational overhead.

Hyperparameter Configuration

Hyperparameters play an important role in determining the efficacy and efficiency of the fine-tuning process. Our configuration can be divided into two main categories: LoRA-specific parameters and general training parameters.

1. LoRA-specific Parameters:

- **Rank (r):** We tested rank values of 4, 16, and 32. Lower ranks (e.g., 4) offer faster training and smaller memory footprint, while higher ranks (e.g., 32) allow for more expressive adaptations.
- **Alpha (α):** We set scaling factors of match the rank in each training test case.

2. General Training Parameters:

- **Learning Rate:**
  - Initial learning rate: 2e-4
  - Scheduler type: Linear
  - Warm-up steps: Five

  The learning rate of 2e-4 was found to be a good balance between fast convergence and stability. We used a linear scheduler to gradually decrease the learning rate over time, which helps in fine-tuning. The warm-up steps help to prevent early training instability by slowly increasing the learning rate from a very small value to our chosen initial rate over the first five steps.
- **Batch Size:**
  - Per device train batch size: 8
  - Gradient accumulation steps: 2

  We chose a small per-device batch size to accommodate GPU memory constraints. However, to benefit from larger effective batch sizes, we used gradient accumulation over 2 steps. This effectively creates a batch size of 16 (8 * 2), allowing for more stable gradient updates without requiring more GPU memory.
- **Training Epochs:** We trained for three epochs.
- **Optimizer:**
  - AdamW with 8-bit optimization
  - Weight decay: 0.01

  We chose AdamW as our optimizer for its ability to handle sparse gradients well. The 8-bit optimization reduces memory usage and speeds up training. A weight decay of 0.01 helps prevent overfitting by penalizing large weights.
- **Random seed:** 1810

Through these hyperparameters and training configurations, we established a fine-tuning process that balances computational efficiency with model performance.

The next chapter details how we built our system, covering the technical stack and implementation choices. We'll examine the code and architecture that transformed our theoretical framework into a working knowledge extraction tool.

5 Implementation

This chapter provides a overview of the implementation of our knowledge extraction system. It details the development environment, encompassing both hardware and software components, as well as the architectural design, data processing pipelines, and integration of various modules. By outlining the practical aspects of our system, this chapter elucidates the steps taken to translate our theoretical framework into a functional and efficient application.

5.1 Development Environment

The development environment constitutes the foundation upon which our knowledge extraction system is built. It encompasses the hardware and software resources that facilitate the design, training, and deployment of the system. This section delves into the specifics of both the hardware and software environments, highlighting the tools and configurations that enable seamless development.

5.1.1 Hardware environment

Our hardware environment is meticulously selected to support the computational demands of training large language models and handling extensive data processing tasks. At the core of our setup is the AMD EPYC 7413 Central Processing Unit (CPU), which boasts 24 cores and 48 threads with a base clock speed of 2.65 GHz and a boost speed of up to 3.6 GHz. This high core count and multi-threading capability are essential for efficient parallel processing, enabling us to handle large-scale data operations and model training tasks with ease.

Complementing the CPU, we used NVIDIA RTX A6000 GPU, equipped with 48 GB of High Bandwidth Memory (VRAM). providing substantial computational power and memory bandwidth optimized for deep learning workloads. The use of multiple GPUs facilitates distributed training, significantly reducing the time required to train large models like LLaMA 3.1 8B.

To ensure smooth data handling and processing, our system is equipped with 64GiB RAM. This ample memory capacity minimizes bottlenecks during model training and inference, allowing for the seamless management of extensive datasets. Additionally, we employ 200 GB of NVMe SSD storage, which offers rapid data access and transfer rates. This high-speed storage is crucial for managing large datasets and reducing latency during both training and evaluation phases.

5.1.2 Software environment

The software environment encompasses the operating systems, programming languages, libraries, frameworks, and tools that constitute the backbone of our knowledge extraction system. The following outlines the key components of our software stack:

- **Operating System:** Ubuntu 24.04 LTS, a Linux-based operating system known for its stability, security, and extensive community support. Ubuntu provides a robust foundation for our development environment, offering compatibility with machine learning tools and libraries.
- **Programming Languages:** Python 3.10 serves as our primary programming language. Python's extensive ecosystem of scientific and machine learning libraries, coupled with its readability and ease of use, makes it an ideal choice for implementing complex algorithms and data processing pipelines.
- **Libraries and Frameworks:**
  - **PyTorch (2.4.0):** An open-source machine learning framework that provides a flexible and efficient platform for deep learning model development. We utilize PyTorch for its dynamic computational graphs and extensive GPU acceleration capabilities.
  - **Unsloth (2024.9.post4):** A specialized library for efficient fine-tuning of Large Language Models (LLMs). It provides optimized training procedures and memory management techniques specifically designed for LLMs.
  - **Transformers (4.46.0):** Hugging Face's transformers library, providing state-of-the-art Natural Language Processing models and tools. We leverage this for implementing and fine-tuning our language models.
  - **Datasets (2.20.0):** Hugging Face's library for efficiently loading and processing large-scale datasets, offering seamless integration with modern machine learning workflows.
  - **NumPy (1.26.3):** A fundamental package for scientific computing in Python, providing support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions.
  - **Pandas (2.2.2):** A powerful data manipulation and analysis library, used for structured data processing and feature engineering in our pipeline.
  - **Tqdm (4.66.4):** A fast, extensible progress bar library for Python, enabling visual feedback during long-running operations and data processing tasks.
  - **PEFT (0.12.0):** Parameter-Efficient Fine-Tuning library, providing efficient methods for adapting large language models with minimal computational overhead.
  - **Sentence-Transformers (3.0.1):** A Python framework for state-of-the-art sentence and text embeddings, utilized for semantic similarity computations and text representation.

- **TRL (0.8.6):** Transformer Reinforcement Learning library, enabling advanced fine-tuning techniques and reinforcement learning approaches for language models.
- **Plotly (5.24.1):** An interactive visualization library producing publication-quality figures and dashboards for result analysis and presentation.
- **NetworkX (3.4.2):** A Python package for creating, manipulating, and studying complex networks and graphs, essential for knowledge graph operations.
- **Neo4j (5.25.0):** A native graph database platform, providing robust storage and querying capabilities for our knowledge graphs, with its Python driver enabling seamless integration.

5.2 Data Pipeline

The data pipeline forms a critical component of our knowledge extraction system, orchestrating data processing and preparation for both model training and inference. At its core lies the **DataPipeline** class, which manages data loading, formatting, and transformation operations.

The pipeline architecture integrates several components that work in concert to convert raw data into a format optimized for language model fine-tuning. The basic class initialization is structured as follows:

```
def __init__(self, tokenizer, json_file_path: str, is_test: bool = False):
    self.tokenizer = tokenizer
    self.json_file_path = json_file_path
    self.eos_token = tokenizer.eos_token
    self.is_test = is_test
    self.dataset = None
```

Listing 5.1: Constructor method initializing the DataPipeline class

The initialization configures essential pipeline components, incorporating the *tokenizer* for text processing and the end-of-sequence token, data source path, and configuration flags. The *is_test* parameter enables dynamic adaptation between training and inference scenarios.

Once initialized, the pipeline's first task is to load and validate the input data instances, where each instance includes text, ontology specifications, and hypergraph representations:

```
def read_json(self) -> List[Dict[str, Any]]:
    try:
        with open(self.json_file_path, 'r', encoding='utf-8') as file:
            data = json.load(file)
        if not isinstance(data, list):
            raise ValueError("JSON data should be a list of instances.")
        return data
    except FileNotFoundError:
        raise FileNotFoundError(f"File not found: {self.json_file_path}")
    except json.JSONDecodeError as e:
        raise ValueError(f"Invalid JSON format: {e}")
```

Listing 5.2: Method to read JSON file and validate it

This loading process not only reads the data but also performs initial validation, ensuring that the data structure meets our requirements before proceeding with further processing.

The pipeline's core functionality emerges in the creation of structured data instances, supporting our dual-task approach to knowledge extraction, where we create instances to

train the model to first understand the domain structure and generate ontology and then extract specific information within that structure.

```python
def create_dataset(self, data: List[Dict[str, Any]]) -> Dataset:
    instructions = []
    inputs = []
    ground_truth_ontologies = []
    ground_truth_hypergraphs = []
    for idx, instance in enumerate(data):
        text = instance['text']
        ontology = json.dumps(instance['ontology'], ensure_ascii=False)
        knowledge_graph = json.dumps(instance['hypergraph'], ensure_ascii=False)
        # Ontology generation instance
        instructions.append("Generate an ontology from the given text.")
        inputs.append(text)
        ground_truth_ontologies.append(ontology)
        ground_truth_hypergraphs.append("")
        # Knowledge graph generation instance
        input_ontology = self.format_ontology(instance['ontology'])
        instructions.append("Generate a knowledge graph based on the provided
            text and ontology.")
        inputs.append(f"Text:\n{text}\n\n{input_ontology}")
        ground_truth_ontologies.append("")
        ground_truth_hypergraphs.append(knowledge_graph)

    dataset_dict = {
            "instruction": instructions,
            "input": inputs,
            "ground_truth_ontology": ground_truth_ontologies,
            "ground_truth_hypergraph": ground_truth_hypergraphs
        }

    dataset = Dataset.from_dict(dataset_dict)
    return dataset
```

Listing 5.3: Method to create a dataset for ontology and knowledge graph generation

This process creates paired instruction-input examples, with each input text generating two distinct training instances. To support this dataset creation, the pipeline includes a ontology formatting method that structures the ontological information in a clear, hierarchical format. This formatting is essential for maintaining consistency in how ontological information is presented to the language model in the hypergraph generating phase.

With the dataset created and ontologies formatted, the pipeline then structures these instances into standardized prompts. Our prompt design is inspired by the Stanford Alpaca project's instruction-tuning format [42], which has demonstrated success in fine-tuning language models for specific tasks. We adapted this format for our knowledge extraction

system, incorporating several key design principles to optimize both model training and inference:

```
prompt_template = (
    "Below is an instruction that describes a task, paired with an input that
        provides further context. "
    "Write a response that appropriately completes the request.\n\n"
    "### Instruction:\n{instruction}\n\n"
    "### Input:\n{input}\n\n"
    "### Response:\n"
    f"{'{output}{eos}' if not self.is_test else ''}"
)
```

Listing 5.4: Prompt template

The template design follows a carefully structured format that enhances the model's ability to understand and process the input data. We use "###" headers to clearly separate different sections of the prompt, providing consistent landmarks that help the model distinguish between instructions, input context, and response areas.

```
def format_prompts(self, examples: Dict[str, List[str]]) -> Dict[str, List[str]]:
    texts = [
        prompt_template.format(
            instruction=inst,
            input=inp,
            output=out if not self.is_test else "",
            eos=self.eos_token if not self.is_test else ""
        )
        for inst, inp, out in zip(
            examples["instruction"],
            examples["input"],
            examples.get("output", [""] * len(examples["instruction"]))
        )
    ]
    return {"text": texts}
```

Listing 5.5: Prompt formatting method

This implementation showcases a key aspect of our data processing strategy: the differential handling of training and inference scenarios. During training, we include both the expected output and an end-of-sequence (EOS) token, which serves as a crucial signal for the model to learn where its generation should stop. This EOS token is particularly important as it helps the model develop a clear understanding of response boundaries, leading to more controlled and accurate generations.

When the pipeline operates in inference mode (*is_test=True*), it omits both the output and EOS token, creating space for the model to generate its responses. This dual-mode functionality allows us to use the same pipeline for both training and inference while main-

taining consistent data formatting. The method also handles cases where outputs might not be available by providing default empty strings, ensuring smooth operation in all scenarios.

To ensure optimal model fine-tuning, the pipeline includes functionality to analyze sequence lengths across the dataset:

```
def calculate_max_seq_length(self, batch_size: int = 1000) -> int:
    if self.dataset is None:
        raise ValueError("Dataset is not prepared. Call ‘prepare()‘ first.")

    max_length = 0
    for i in range(0, len(self.dataset), batch_size):
        batch = self.dataset[i:i + batch_size]['text']
        tokenized = self.tokenizer(batch, truncation=False,
            add_special_tokens=False)
        lengths = [len(ids) for ids in tokenized['input_ids']]
        batch_max = max(lengths)
        max_length = max(max_length, batch_max)
```

Listing 5.6: Sequence length analysis method

This sequence length analysis plays a vital role in ensuring stable and efficient model fine-tuning. By processing the dataset in manageable batches, we can accurately determine the maximum sequence length needed for our model configuration. This information is crucial for several aspects of the training process: it allows us to properly set the model's maximum position embeddings, configure the attention mechanism parameters, and optimize memory usage during training.

The entire pipeline comes together in a straightforward usage pattern:

```
# Initialize and run the pipeline
pipeline = DataPipeline(
    tokenizer=tokenizer,
    json_file_path="training_data.json",
    is_test=False
)
dataset = pipeline.prepare()
max_length = pipeline.calculate_max_seq_length()
```

Listing 5.7: Example pipeline usage

5.3 Model Fine-Tuning

Our model fine-tuning implementation leverages the Unsloth library, a specialized framework designed for optimizing large language model training and fine-tuning. Our implementation employs Parameter-Efficient Fine-Tuning (PEFT), specifically focusing on LoRA (Low-Rank Adaptation), to efficiently adapt the base model to our knowledge extraction tasks, for more details look at section 2.4 and 4.2.2

The fine-tuning process begins with loading the pre-trained base model (not the instruction-tuned version) using Unsloth's FastLanguageModel:

```
from unsloth import FastLanguageModel

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "unsloth/Meta-Llama-3.1-8B",
    max_seq_length = max_seq_length,
    dtype = None,
    load_in_4bit = True,
)
```

Listing 5.8: Model initialization using 4-bit quantization

This initialization step uses 4-bit quantization, a technique that reduces the precision of model weights from 32-bit floating-point numbers to 4-bit integers (see section 2.2.3). This compression significantly reduces memory usage while maintaining most of the model's performance. The *max_seq_length* parameter, determined by our data pipeline's sequence length analysis, ensures the model can handle our longest training sequences without truncation.

The PEFT model is then configured using LoRA, which adds trainable rank decomposition matrices to key transformer layers:

```
model = FastLanguageModel.get_peft_model(
    model,
    r = 16,                           # LoRA rank
    lora_alpha = 16,                  # LoRA scaling factor
    lora_dropout = 0,                 # Dropout probability for LoRA layers
    target_modules = [                # Transformer layers to apply LoRA
        "q_proj", "k_proj", "v_proj", "o_proj",
        "gate_proj", "up_proj", "down_proj"
    ],
    bias = "none",                    # Bias configuration
    use_gradient_checkpointing = True, # Memory optimization for long sequences
    random_state = 1810,              # Reproducibility seed
)
```

Listing 5.9: LoRA configuration

This configuration step establishes the architecture for efficient fine-tuning through several key parameters:

- The rank parameter ***r*** determines the complexity of our learned adaptations. A higher rank allows for more expressive adaptations but requires more memory and computation.
- ***lora_alpha*** controls the magnitude of LoRA adaptations relative to the original weights and is typically set to match the rank.
- ***target_modules*** specifies which transformer components receive LoRA adaptations. We target all key projection matrices in the attention mechanism (*q_proj, k_proj, v_proj, o_proj*) and MLP layers (*gate_proj, up_proj, down_proj*).

Building upon our efficient model adaptation configurations, we next configure the training process itself. This configuration set the hyperparameters to optimize learning dynamics with computational resources:

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    per_device_train_batch_size=8,    # Batch size per GPU
    gradient_accumulation_steps=2,     # Effective batch size multiplier
    warmup_steps=5,                    # Learning rate warm-up period
    num_train_epochs=3,                # Total number of training epochs
    learning_rate=2e-4,                # Initial learning rate
    fp16=not is_bfloat16_supported(), # Precision based on hardware support
    bf16=is_bfloat16_supported(),     # BFloat16 when available
    logging_steps=1,                   # Frequency of metric logging
    optim="adamw_8bit",               # 8-bit Adam optimizer for memory efficiency
    weight_decay=0.01,                # L2 regularization factor
    lr_scheduler_type="linear",        # Learning rate decay schedule
    seed=1810,                         # Random seed for reproducibility
    output_dir="outputs",             # Checkpoint and artifact directory
    save_steps=100,                   # Checkpoint frequency
    save_strategy="steps",            # Checkpoint timing strategy
    save_total_limit=10,              # Maximum saved checkpoints
    report_to="wandb",               # Integration with Weights & Biases for
        experiment tracking
    run_name="finetuning-knowledge-extraction", # Experiment identifier
)
```

Listing 5.10: Training arguments configuration

For further explanations of the hyperparameters, refer to Section 4.3.3.

With the model and training configurations in place, we proceed to the training phase. The training process involves initializing a trainer object and invoking the training routine.

```
from trl import SFTTrainer

trainer = SFTTrainer(
    model=model,
    tokenizer=tokenizer,
    train_dataset=prepared_dataset,
    dataset_text_field="text",
    max_seq_length=max_seq_length,
    dataset_num_proc=2,              # Number of preprocessing workers
    training_args = training_args)

trainer.train()
```

Upon invoking *trainer.train()*, the fine-tuning process commences. The trainer handles the training loop, including forward and backward passes, optimizer steps, learning rate scheduling, and checkpointing. Additionally, integration with Weights & Biases (wandb) facilitates real-time monitoring of training metrics such as loss, learning rate, and validation performance

5.4 Model Evaluation

After fine-tuning our language model, it is imperative to rigorously evaluate its performance to ensure that it meets the desired objectives for knowledge extraction. This section details the implementation of our model evaluation pipeline, which leverages the previously established data processing and model fine-tuning pipelines. The evaluation process encompasses data preparation, model inference, and result aggregation, culminating in the generation of evaluation metrics.

The evaluation process begins by initializing the tokenizer and preparing the testing dataset. This ensures that the model receives input data in the same format as it was during training, facilitating consistent and reliable evaluations.

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained('unsloth/Meta-Llama-3.1-8B')
# Initialize DataPipeline with the tokenizer and path to testing data
data_pipeline = DataPipeline(
    tokenizer=tokenizer,
    json_file_path="testing_data.json",
    is_test=True)
# Prepare the dataset and determine sequence length requirements
test_dataset = data_pipeline.prepare()
max_length = data_pipeline.calculate_max_seq_length()
```

Listing 5.11: DataPipeline Initialization for Evaluation

This setup mirrors our training configuration while switching to evaluation mode (*is_test=True*), ensuring our test data receives consistent preprocessing and omits both the ground truth output and EOS token, creating space for the model to generate its responses.

With the testing dataset prepared, the next step involves loading the fine-tuned model configured for inference. This model incorporates the previously trained LoRA adapter parameters for knowledge extraction.

```
from unsloth import FastLanguageModel

# Load the fine-tuned model with LoRA adaptations
model, _ = FastLanguageModel.from_pretrained(
    model_name="path_to_the_lora_files",
    max_seq_length=max_length,
    dtype=None,
    load_in_4bit=True,
)

# Switch the model to inference mode
FastLanguageModel.for_inference(model)
```

Listing 5.12: DataPipeline Initialization for Evaluation

The *FastLanguageModel.from_pretrained()* method is utilized to load the fine-tuned model from the specified path containing the LoRA adaptation files. The parameter load_in_4bit=True applies 4-bit quantization as by the training. After loading, the model and merge it with the LoRA adapter, it is switched to inference mode using *FastLanguageModel.for_inference(model)*.

After setting the model to inference mode, we iterated through the test dataset, generating model outputs, and aggregating the results for subsequent analysis. The foundation of our evaluation loop lies in carefully processing each test instance and preparing it for model inference:

```
with torch.no_grad():
    for idx in tqdm(range(len(test_dataset)), desc="Generating outputs"):
        instance = test_dataset[idx]
        instruction = instance['instruction']
        input_text = instance['text']

        ground_truth_ontology = instance['ground_truth_ontology']
        ground_truth_hypergraph = instance['ground_truth_hypergraph']
```

Listing 5.13: Evaluation loop – Instance retrieval

This initial processing stage extracts all necessary components from each test instance. The *torch.no_grad()* context ensures memory efficiency by disabling gradient calculations during inference, while the progress bar provided by *tqdm* offers real-time feedback on the evaluation progress. Each instance contains both the input text and its corresponding

ground truth annotations for both ontology and knowledge graph tasks, maintaining the dual-task structure established in our training pipeline.

The extracted instance data undergoes tokenization before being passed to the model:

```
inputs = tokenizer(
            [input_text],
            return_tensors="pt",
        ).to('cuda')

generated_ids = model.generate(
    **inputs,
    max_new_tokens=8000,
    use_cache=True
)
```

Listing 5.14: Evaluation loop – Input tokenization and output generation process.

GPU acceleration and generation caching optimize inference speed, while the generous *max_new_tokens* allowance ensures complete output generation. This maintains consistency with our training pipeline while prioritizing generation completeness.

Once the model generates its output, we extract and process the generated content:

```
input_length = inputs['input_ids'].shape[1]
generated_only_ids = generated_ids[:, input_length:]
generated_text = tokenizer.batch_decode(
    generated_only_ids,
    skip_special_tokens=True
)[0].strip()
```

Listing 5.15: Evaluation loop – Extraction and decoding of generated outputs.

By slicing away input tokens and removing special markers, we obtain clean output text ready for task-specific processing, ensuring accurate evaluation of the model's generations.

The evaluation process differentiates between our dual tasks, managing outputs according to their specific requirements:

```
if "Generate an ontology" in instruction:
            output_ontology = generated_text
            current_text = input_text
            current_ground_truth_ontology = ground_truth_ontology
elif "Generate a knowledge graph" in instruction:
    output_hypergraph = generated_text
    current_ground_truth_hypergraph = ground_truth_hypergraph
```

Listing 5.16: Evaluation loop – Dual-task output routing and context preservation.

This stage maintains the relationship between generated outputs and their corresponding tasks, ensuring proper pairing of ontology and knowledge graph generations. By examining the instruction associated with each instance, we route the generated content to the appropriate output variable, maintaining the structural integrity of our dual-task evaluation framework.

The final stage of our evaluation loop captures the complete context of each generation task:

```
result = {
    "text": current_text,
    "ground_truth_ontology": current_ground_truth_ontology,
    "ground_truth_hypergraph": current_ground_truth_hypergraph,
    "output_ontology": output_ontology,
    "output_hypergraph": output_hypergraph
}

with open(jsonl_file_path, 'a', encoding='utf-8') as f:
    f.write(json.dumps(result, ensure_ascii=False) + '\n')
```

Listing 5.17: Evaluation loop – Result recording.

With the results captured and stored, we can now apply the evaluation metrics established earlier in Section 4.3.2 to assess the model's performance against the ground truth annotations.

5.5 Knowledge Integration

The successful extraction of knowledge from unstructured text requires a method for storing and organizing the identified relations and entities. Our knowledge integration system implements a graph-based storage solution that captures both traditional relationships and complex multi-entity interactions. By leveraging Neo4j, a native graph database, we create an interconnected knowledge network that preserves the rich semantic structures discovered during the extraction process while enabling efficient querying and knowledge retrieval.

The integration process begins with establishing a secure connection to our Neo4j database:

```
from neo4j import GraphDatabase
# Neo4j connection configuration
uri = "bolt://your-neo4j-domain:port"
username = "username"
password = "password"
driver = GraphDatabase.driver(uri, auth=(username, password))
```

Listing 5.18: Database connection setup.

This connection setup establishes a persistent driver that manages our database interactions efficiently, using the Bolt protocol for optimized communication with the Neo4j server.

Our first integration task handles the creation of entity nodes, preserving the type information and attributes extracted by our model:

```
def create_nodes(tx, nodes):
    for node in nodes:
        query = f"""
        MERGE (n {{id: $id}})
        SET n:{node['type']}
        SET n.id = $id
        SET n += $attributes
        """
        tx.run(query, id=node['id'], attributes=node['attributes'])
```

Listing 5.19: Node creation implementation preserving entity attributes and types.

This implementation uses Neo4j's MERGE operation to ensure idempotency, preventing duplicate nodes while allowing updates to existing entities. The type information from our ontology is preserved through node labels, while extracted attributes are stored as node properties.

Following node creation, we establish the relationships identified during knowledge extraction:

```
def create_edges(tx, edges):
    for edge in edges:
        query = f"""
        MATCH (source {{id: $source}})
        MATCH (target {{id: $target}})
        MERGE (source)-[r:{edge['label']}]->(target)
        SET r += $attributes
        """
        tx.run(query, source=edge['source'], target=edge['target'],
               attributes=edge['attributes'])
```

Listing 5.20: Edge creation preserving relationship types and attributes.

The relationship integration carefully matches source and target nodes before establishing connections, using MERGE to handle potential relationship updates. This ensures our knowledge graph maintains consistency while accommodating new information.

Following basic node and edge integration, our implementation extends to handling hyperedges, which capture complex multi-entity relationships that go beyond simple binary connections:

```
def create_hyperedges(tx, hyperedges):
    tx.run("""
    UNWIND $hyperedges AS hyperedge
    CREATE (h:Hyperedge {type: hyperedge.type})
    SET h += hyperedge.attributes
    WITH h, hyperedge
    UNWIND hyperedge.nodes AS nodeId
    MATCH (n {id: nodeId})
    CREATE (n)-[:PART_OF_HYPEREDGE]->(h)
    """, hyperedges=hyperedges)
```

Listing 5.21: Hyperedge implementation capturing complex multi-entity relationships.

This implementation represents hyperedges as special nodes connected to their participants through standardized relationships, enabling efficient querying of complex relationship patterns while maintaining the semantic richness of our extracted knowledge.

Through this integration implementation, we transform our model's extraction outputs into a rich, queryable knowledge graph that preserves all semantic relationships identified during the extraction process. The resulting graph database provides a foundation for knowledge exploration and downstream applications, completing our pipeline from unstructured text to structured, accessible knowledge.

6 Experiments and Results

In this chapter, we present and analyze the experimental results of our study, examining the effectiveness of adapting Large Language Models (LLMs) for knowledge extraction tasks. The experiments and their outcomes are evaluated based on the metrics, testing dataset, and training parameters detailed in the previous chapter.

6.1 Experiments scenario

Our experimental investigation focused on exploring the adaptability of LLMs, specifically Llama 3.1 8B, for complex knowledge extraction tasks. We implemented a joint learning approach where the model was simultaneously trained on two interconnected tasks:

1. Generating ontologies from given text inputs
2. Extracting relevant information based on both the input text and the generated ontology

A key aspect of our experimental design was the implementation of multi-level information extraction in a single pass, encompassing:

1. Named entities and their attributes
2. Binary relations and their attributes
3. Hyperedges and their attributes

This comprehensive approach represents a significant challenge in the field of knowledge extraction, as it requires the model to understand and process multiple layers of semantic information simultaneously.

Our experimental work was driven by the hypothesis that LLMs possess extensive knowledge across diverse domains and languages, making them potentially powerful universal knowledge extraction tools when properly adapted. To test this hypothesis, we employed the LoRA (Low-Rank Adaptation) fine-tuning approach, conducting two distinct sets of experiments.

6.1.1 First Experimental Set: Full Block Adaptation

In the initial experimental phase, we implemented a comprehensive adaptation strategy targeting all transformer block layers within the Llama 3.1 8B model. This approach involved systematically modifying the model's parameter space through LoRA matrices of varying ranks. We experimented with three distinct matrix configurations: rank-4, rank-16, and rank-32 matrices. The rank-4 configuration represented our most parameter-efficient approach, introducing minimal additional trainable parameters while potentially capturing essential task-specific features. The rank-16 configuration offered a balanced trade-off between model complexity and adaptability, while the rank-32 configuration provided the highest degree of freedom in parameter adjustment, potentially allowing for more nuanced feature learning at the cost of increased computational overhead.

6.1.2 Second Experimental Set: Selective Attention Layer Adaptation

Building upon insights from our initial experiments, the second phase employed a more targeted approach by exclusively focusing on the attention mechanisms within the transformer architecture. This selective adaptation strategy was motivated by the hypothesis that attention layers play a crucial role in capturing complex relationships necessary for knowledge extraction tasks. We maintained the same rank variations (4, 16, and 32) for the LoRA matrices but concentrated their application solely on the attention computation components. This focused approach aimed to investigate whether comparable or superior performance could be achieved with a more economical parameter budget by targeting specifically the model's attention mechanisms. The selective adaptation strategy potentially offers advantages in terms of computational efficiency and training stability, while still maintaining the model's capacity to learn task-specific knowledge extraction patterns.

6.2 Results

In this section, we present an analysis of our experimental outcomes across six distinct test cases, divided equally between our two experimental sets. For each test case, we examine the training dynamics through loss curves and evaluate the model's performance using our three primary metrics: Structure Consistency, Knowledge Similarity, and Ontology Adherence. This systematic approach allows us to thoroughly assess the effectiveness of different LoRA configurations and adaptation strategies. Sample outputs can be found in the Appendix

6.2.1 Model Parameters and Adapter Configurations

Before presenting the detailed results, we first analyze the parameter space and computational requirements of each configuration. Table 6.2.1 presents a comprehensive overview of the parameter counts and adapter sizes for each experimental configuration.

Adapter Type	LoRA Rank (r)	Trainable Params.	Affected Params. (%)	Size (MB)
Full-Block	4	10,485,760	0.13%	40.1
Full-Block	16	41,943,040	0.52%	160.1
Full-Block	32	83,886,080	1.04%	320.1
Attention-Only	4	3,407,872	0.04%	13
Attention-Only	16	13,631,488	0.16%	52
Attention-Only	32	27,262,976	0.33%	104

Table 6.1: Analysis of parameter efficiency in different LoRA configurations

The results in Table 6.2.1 reveal remarkable parameter efficiency across different LoRA configurations. Full-Block adapters with rank 4 affect only 0.13% of the model's parameters while requiring 40.1MB of storage, demonstrating significant efficiency compared to traditional fine-tuning approaches. This efficiency becomes even more pronounced with Attention-Only adapters, which affect a mere 0.04% of parameters at rank 4 while requiring just 13MB of storage space.

The relationship between rank selection and parameter count shows a linear progression, with higher ranks naturally demanding more resources. At rank 32, Full-Block adapters affect 1.04% of parameters with a 320.1MB storage requirement, while Attention-Only adapters maintain efficiency by affecting only 0.33% of parameters and requiring 104MB. This difference highlights how Attention-Only adapters consistently achieve greater parameter efficiency, requiring approximately one-third of the resources needed by Full-Block configurations at the same rank.

The storage requirements for all configurations remain practical for deployment scenarios, ranging from 13MB to 320.1MB. Even the most parameter-intensive configuration maintains a relatively small footprint compared to full model fine-tuning, making LoRA adapters a viable solution for resource-constrained environments.

6.2.2 Full Block Adaptation Results

For each rank configuration in the full block adaptation test case, we analyzed the training dynamics and performance metrics. Figure 6.1 shows the training loss curves for all three rank configurations (4, 16, 32).

The training dynamics across different LoRA adapter configurations demonstrate several distinct phases of convergence and performance characteristics, offering valuable insights into the relationship between rank selection and model optimization. Through careful analysis of the entropy loss curves, we observe three primary phases of learning behavior that characterize the adaptation process.

In the initial phase (steps 0-50), as illustrated in Figure 6.1, all configurations exhibit rapid convergence from their starting loss values of 1.2. This swift adaptation demonstrates the inherent capability of LLMs to leverage their pre-trained knowledge while adapting to specialized tasks, even when modifying only a small fraction of their parame-

ters through low-rank updates. The consistent sharp descent across all rank configurations suggests that the model efficiently transfers its fundamental language understanding to the target task.

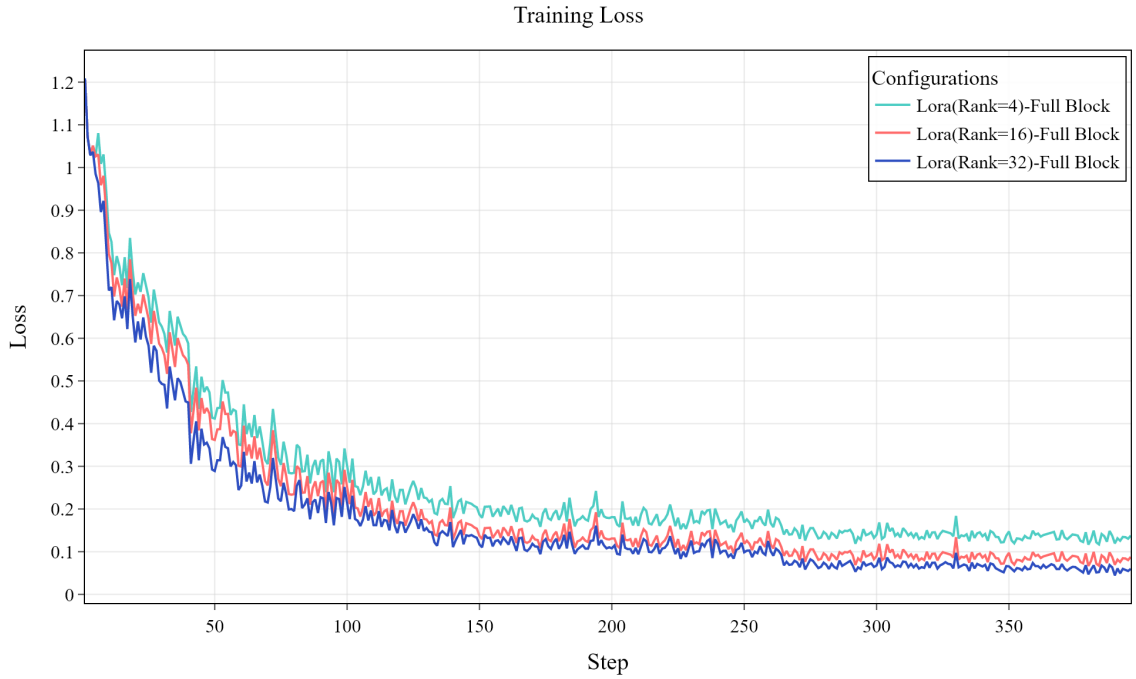


Figure 6.1: Training loss curves in full block adaptation experiments.

The intermediate phase (steps 50-200) reveals a transition to more gradual learning trajectories, where higher-rank configurations (rank-16 and rank-32) begin to demonstrate marginally superior performance compared to their lower-rank counterpart. This phase highlights the emerging benefits of increased parameter capacity, though with diminishing returns between rank-16 and rank-32 configurations.

During the final phase (steps 200-350), the models exhibit convergence characteristics typical of well-trained systems. The rank-32 configuration achieves optimal performance with final loss values around 0.05, closely followed by the rank-16 configuration with loss around 0.08. The rank-4 configuration, while demonstrating stable convergence, plateaus at a notably higher loss value of approximately 0.15.

These observations carry significant implications for practical implementations of LoRA adaptations. The marginal performance improvement between rank-16 and rank-32 configurations suggests an efficiency threshold, indicating that increasing rank beyond 16 may not justify the additional computational overhead for this particular application. Importantly, the relatively small fluctuations in the loss curves across all configurations indicate stable training dynamics, suggesting effective learning rate scheduling and robust optimization. Furthermore, the rank-4 configuration, despite its higher final loss values, presents an interesting trade-off between model efficiency and performance, potentially suitable for applications where computational efficiency takes precedence over optimal performance.

Our evaluation of the full block adaptation performance reveals nuanced patterns across different metrics and element types, as shown in Table 6.2.2 and Figures 6.2 and 6.3.

The structural validation results demonstrate the superior reliability of higher-rank configurations, with both rank-16 and rank-32 achieving near-perfect syntax and schema validity (99-100%), while rank-4 maintains respectable performance (93-97%) despite its reduced parameter space.

Category	**Metric**	**Rank-4**		**Rank-16**		**Rank-32**	
		Ont.	**Hyp.**	**Ont.**	**Hyp.**	**Ont.**	**Hyp.**
Structural Validation	Syntax Validity (%)	94%	97%	100%	100%	100%	99%
	Schema Validity (%)	93%	94%	100%	100%	100%	99%
Knowledge Similarity	Exact Match (%)	92%	92%	97%	94%	98%	95%
	Sequence Matcher (%)	93%	95%	98%	97%	98%	97%
	Semantic Similarity (%)	93%	96%	98%	98%	98%	98%
Ontology Adherence	Error Rate (%)	9.72%		2.40%		2.39%	

Table 6.2: Full Block Adaptation – Performance metrics

The knowledge similarity metrics in Table 6.2.2 demonstrate a consistent pattern of high performance across different matching criteria, with each successive matching type showing incremental improvements. In exact matching, rank-32 configurations achieve optimal performance (98% for ontology, 95% for hypergraph elements), with rank-16 following closely (97% and 94% respectively) and rank-4 maintaining performance at 92%. The sequence matcher results show improved performance across all configurations, with rank-16 and rank-32 achieving 97-98% similarity for both ontology and hypergraph elements. Semantic similarity scores demonstrate the highest performance, reaching 98% for higher ranks across both ontology and hypergraph elements, indicating that essential meaning and relationships are preserved even when syntactic details might vary. This progression suggests that while models maintain high fidelity in exact structural matching, they excel even further at preserving semantic relationships and sequential patterns in the knowledge structure.

The ontology adherence error rates show an improvement when moving from rank-4 (9.72%) to rank-16 (2.40%), but only marginal gains with rank-32 (2.39%). This pattern of diminishing returns provides strong evidence for the practical sufficiency of rank-16 configurations in our case.

The element-specific analysis, illustrated in Figures 6.2 and 6.3, reveals consistent patterns across all elements in both tasks ontology and hypergraph generation. The ontology elements analysis, depicted in Figure 6.2, entity type similarity shows consistency across all matching types, with rank-16 and rank-32 configurations achieving above 98% similarity, while rank-4 maintains strong performance above 93%. Relation type similarity exhibits a similar pattern but with slightly lower absolute values, particularly for rank-4

configurations (91-93%). This suggests that relationship patterns are more challenging to capture with lower-rank adaptations. The hyperedge type similarity demonstrates robust preservation of complex structural relationships, with all ranks showing strong performance (>90%) and particularly impressive results for semantic matching. Attribute similarity in the ontology context shows high consistency across ranks, with even rank-4 configurations maintaining scores above 90%, indicating that basic property relationships are well-preserved even with limited parameter capacity.

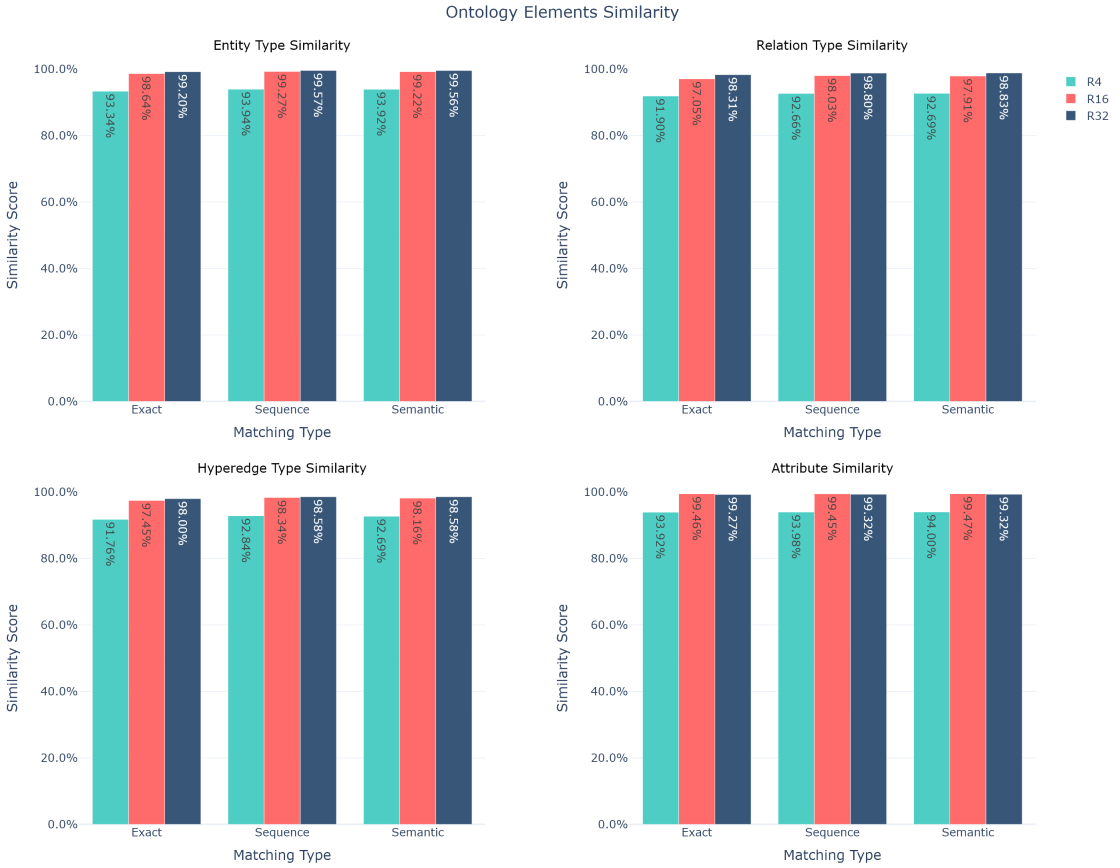


Figure 6.2: Full Block Adaptation – Ontology elements similarity

For hypergraph elements (Figure 6.3), node similarity metrics demonstrate strong performance across all configurations, with rank-16 and rank-32 achieving particularly high scores (>97%) in semantic matching. Edge similarity shows more pronounced differences between rank configurations, with rank-32 maintaining consistent performance above 96% across all matching types, while rank-4 shows more variation (92-95%). The hyperedge similarity metrics reveal interesting patterns in the preservation of complex relationships, where even rank-4 configurations maintain similarity scores above 90%, suggesting that the essential structure of higher-order relationships is preserved even with limited rank.

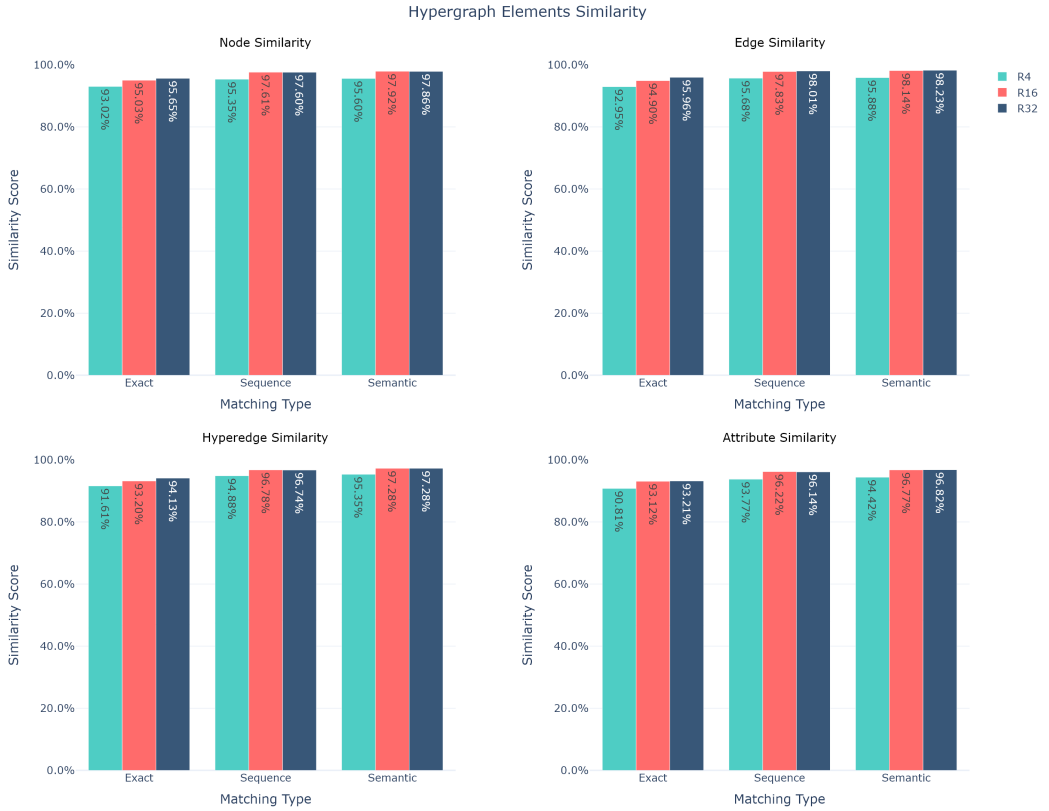


Figure 6.3: Full Block Adaptation – Hypergraph elements similarity

6.2.3 Attention-Only Adaptation Results

The attention-only adaptation experiments revealed interesting patterns in performance efficiency. Figure 6.4 illustrates the training dynamics for these configurations.

The training dynamics across different LoRA adapter configurations with attention-only tuning demonstrate several notable patterns, particularly when compared to full-block adaptation. The analysis reveals interesting insights about the efficiency of targeting solely the attention mechanisms in large language models.

In the initial phase (steps 0-50), all attention-only configurations show rapid convergence from their starting loss values of approximately 1.0-1.2, similar to their full-block counterparts. However, a key distinction emerges in the magnitude of fluctuations, where attention-only configurations exhibit more pronounced oscillations during this early learning phase. This suggests that while attention mechanisms can effectively adapt to the task, the restricted parameter space may result in less stable initial optimization.

The intermediate phase (steps 50-200) reveals a more gradual descent with notably higher volatility compared to full-block tuning. The rank-4 attention-only configuration, in particular, shows significantly larger fluctuations and slower convergence, stabilizing around 0.3-0.4 loss compared to approximately 0.15 in full-block tuning. This increased volatility and higher final loss values indicate that attention-only adaptation, while effective, may not capture task-specific patterns as comprehensively as full-block tuning.

During the final phase (steps 200-350), we observe distinct convergence behaviors across rank configurations. The rank-32 attention-only model achieves the best performance with loss values stabilizing around 0.15-0.2, notably higher than its full-block counterpart (0.05-0.07). This performance gap, combined with increased fluctuations in the loss curves, suggests that while attention mechanisms play a crucial role in model adaptation, the exclusion of feed-forward network updates may limit the model's ability to achieve optimal task-specific performance.

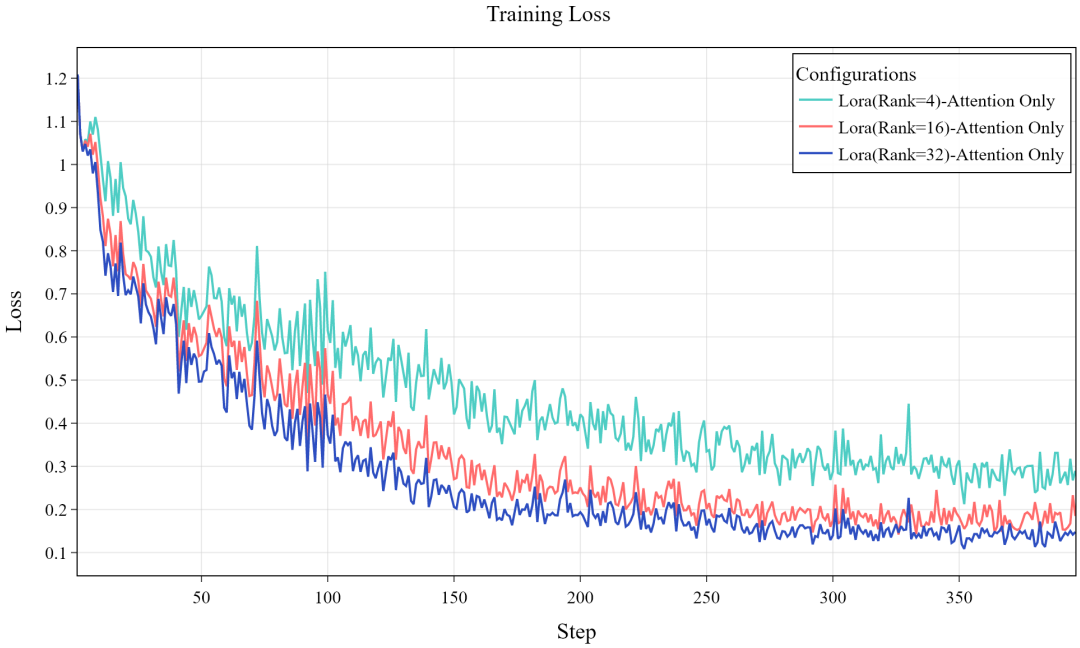


Figure 6.4: Training loss curves in attention only adaptation experiments.

These observations highlight the trade-off between parameter efficiency and model performance in LoRA implementations. While attention-only tuning significantly reduces the number of modified parameters compared to full-block adaptation, it comes at the cost of higher final loss values and increased training volatility. This finding is particularly relevant for practitioners seeking to balance computational efficiency with task performance.

Category	**Metric**	**Rank-4**		**Rank-16**		**Rank-32**	
		Ont.	**Hyp.**	**Ont.**	**Hyp.**	**Ont.**	**Hyp.**
Structural Validation	Syntax Validity (%)	86%	89%	96%	95%	100%	98%
	Schema Validity (%)	83%	87%	91%	95%	100%	97%
Knowledge Similarity	Exact Match (%)	58%	67%	78%	77%	90%	83%
	Sequence Matcher (%)	68%	78%	83%	87%	93%	91%
	Semantic Similarity (%)	59%	69%	83%	88%	94%	92%
Ontology Adherence	Error Rate (%)	43.83%		19.64%		8.82%	

Table 6.3: Attention Only Adaptation – Performance metrics

The attention-only adaptation experiments reveal distinct patterns in performance across different rank configurations, offering insights into the capabilities and limitations of modifying only the attention mechanisms. Our evaluation of the adaptation performance demonstrates systematic variations across metrics and element types, as shown in Table 6.3 and Figures 6.5 and 6.6. The structural validation results indicate a stronger dependence on rank configuration compared to full block adaptation, with rank-32 maintaining strong performance (98-100% for both syntax and schema validity), while rank-16 shows moderate degradation (91-96%) and rank-4 exhibits more substantial drops in performance (83-89%). This pattern suggests that attention mechanisms alone may struggle to maintain structural consistency without sufficient parameter capacity.

The knowledge similarity metrics reveal a nuanced hierarchy of performance across different matching criteria. In exact matching, rank-32 configurations achieve respectable performance (90% for ontology, 83% for hypergraph elements), with rank-16 showing moderate performance (78% and 77% respectively) and rank-4 demonstrating significant degradation (58% and 67%). The sequence matcher results show consistent improvement across all configurations, with rank-32 achieving 91-93% similarity and rank-4 improving to 68-78%, indicating that sequential patterns are better preserved than exact matches. Semantic similarity scores demonstrate similar trends, with rank-32 reaching 92-94% and rank-4 maintaining 59-69%, suggesting that essential meaning preservation requires substantial attention parameter capacity.

The ontology adherence error rates reveal a more pronounced sensitivity to rank selection, with rank-4 showing substantial errors (43.83%), rank-16 demonstrating moderate improvement (19.64%), and rank-32 achieving more acceptable performance (8.82%). This clear progression suggests that attention mechanism capacity plays a crucial role in maintaining ontological consistency, though not achieving the same level of performance as full block adaptation.

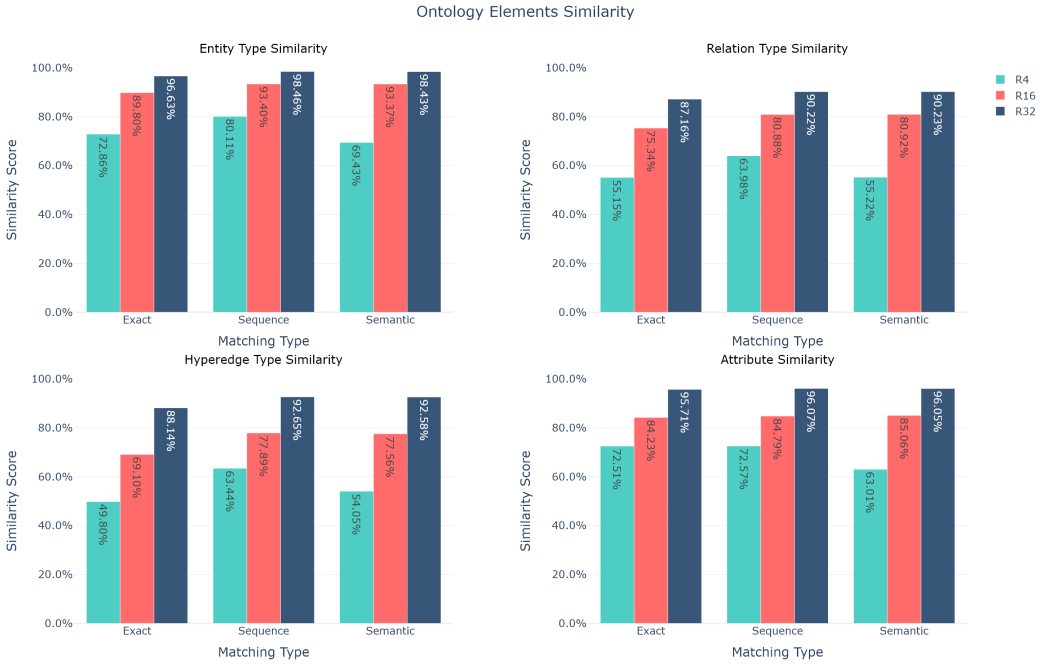


Figure 6.5: Attention Only Adaptation – Ontology elements similarity

The element-specific analysis, illustrated in Figures 6.5 and 6.6, reveals systematic patterns across different element types. For ontology elements, entity type similarity demonstrates relatively robust performance across ranks, with rank-32 maintaining similarity above 90% across all matching types, while rank-4 shows moderate degradation (72-80%). Relation type similarity exhibits more sensitivity to rank selection, with rank-32 achieving 85-90% similarity while rank-4 struggles to maintain consistent performance (55-65%). Hyperedge type similarity reveals interesting patterns in complex relationship preservation, where even rank-4 configurations maintain moderate performance (60-70%). Attribute similarity shows consistent patterns within ranks but clear stratification between them, suggesting that attention mechanisms can maintain basic property relationships with sufficient capacity.

For hypergraph elements, node similarity metrics demonstrate strong rank-dependent performance, with rank-32 maintaining 85-92% similarity across matching types, while rank-4 shows moderate degradation but maintains functional performance (65-75%). Edge similarity reveals more pronounced differences between ranks, particularly in exact matching, where rank-32 achieves 82-92% similarity while rank-4 struggles to maintain consistent performance (64-77%). The hyperedge similarity metrics show systematic variation across ranks, with rank-32 maintaining strong performance (85-92%) while rank-4 demonstrates moderate degradation (65-70%).

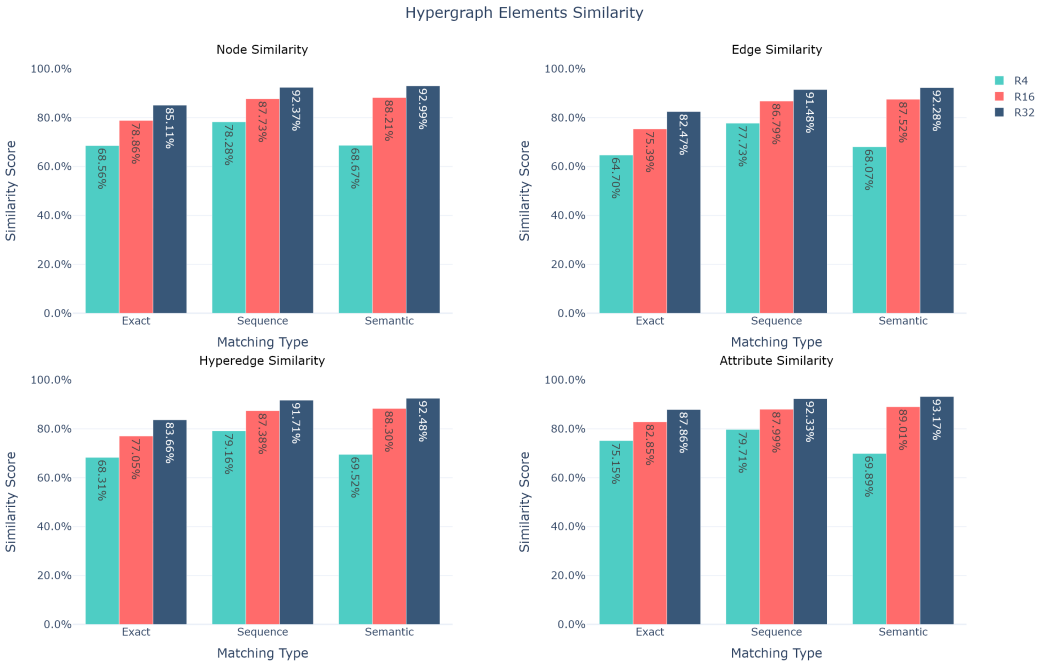


Figure 6.6: Attention Only Adaptation – Hypergraph elements similarity

These observations carry significant implications for practical implementations of attention-only LoRA adaptations. The substantial performance gap between rank configurations suggests that attention-only tuning requires higher ranks to achieve acceptable performance compared to full block adaptation. The relatively larger fluctuations in performance across different element types and metrics indicate that attention mechanisms alone may

not provide the same level of robust adaptation as full block tuning. However, the consistent improvement patterns across ranks suggest that attention-only adaptation can be a viable approach when computational efficiency is prioritized over optimal performance, particularly with higher rank configurations.

Furthermore, the clear stratification of performance across ranks and metrics provides valuable guidance for practitioners. While rank-32 configurations generally maintain acceptable performance across most metrics, the substantial degradation in lower ranks suggests that attention-only adaptation may require careful consideration of rank selection based on specific task requirements and performance thresholds.

7 Conclusion and Future Work

This chapter synthesizes our research findings and contributions in the field of knowledge extraction and Large Language Models (LLMs). Through a systematic investigation of LLMs capabilities in transforming unstructured text into structured, ontology-guided knowledge representations, we have addressed fundamental challenges in automated knowledge extraction. Our research was guided by the central question:

Research question

How can large language models be effectively adapted to extract structured knowledge from unstructured text across various domains?

The investigation of this question was driven by three primary objectives:

1. Developing efficient adaptation techniques for universal schema-guided knowledge extraction system
2. Evaluating LLMs effectiveness in extracting structured knowledge from unstructured text
3. Creating mechanisms for integrating extracted information into coherent knowledge structures

This chapter first presents our conclusions and key findings, followed by a examination of the current limitations, and concludes with promising directions for future research that could address these limitations while advancing the field further.

7.1 Conclusion

This thesis has presented an approach to structured knowledge extraction that uses the capabilities of Language Models through a dual-task framework. The experimental results presented in Chapter 6 demonstrate the effectiveness of adapting LLMs for this task across various domains. The key innovation lies in the synergistic combination of ontology generation and knowledge extraction, demonstrating that LLMs can effectively understand, structure, and extract complex information from unstructured text while maintaining ontological consistency.

Our research underscores that adapted LLMs, even those with relatively modest parameter sizes such as the 8B-parameter LLaMA 3.1, possess the capacity to learn and execute joint, multi-level tasks. These tasks encompass ontology generation, Named Entity Recognition

(NER), Relationship Extraction (RE), and hyperedge extraction with associated attributes and requisite structural frameworks. This capability is a testament to the versatility and adaptability of LLMs when fine-tuned with appropriate techniques like Low-Rank Adaptation (LoRA).

Our experiments explored two distinct adaptation strategies: full transformer block adaptation and attention-only adaptation, each tested with LoRA rank configurations of 4, 16, and 32. The full block approach, which modifies both attention and feed-forward layers, demonstrated superior performance across all evaluation metrics while maintaining remarkable parameter efficiency. Even at lower ranks, the full block strategy showed robust performance, modifying just 0.13% of parameters at rank-4 and scaling to 1.04% at rank-32. In contrast, the attention-only strategy achieved better parameter efficiency, affecting only 0.04% to 0.33% of parameters across ranks, but showed some performance degradation, particularly in handling complex relationships and maintaining ontological consistency. This trade-off suggests that while attention mechanisms play a crucial role in knowledge extraction, the feed-forward layers contribute significantly to the model's ability to process and structure complex information, especially when dealing with intricate knowledge structures and relationships.

This investigation into adaptation strategies led to an insight that the optimal rank selection for LoRA adaptation should be guided by task complexity rather than blindly pursuing higher ranks. In our knowledge extraction tasks, rank-16 configurations emerged as a sweet spot, offering an excellent balance between computational efficiency and performance. While higher ranks showed some improvements, the marginal gains beyond rank-16 were modest relative to the increased computational overhead, suggesting that practitioners should carefully consider the cost-benefit trade-off when selecting rank configurations for similar applications.

The practical implications of these findings became evident in the system's consistent generation of valid JSON structures, particularly with higher rank configurations. This capability demonstrates that properly adapted LLMs can transcend traditional chatbot-style interactions and function as reliable components within larger software systems. The high structural consistency of the output, combined with strict schema adherence, enables direct integration with downstream applications, databases, and other information processing systems without the need for manual intervention.

To address broader challenges in LLM applications, our dual-task framework and ontology-guided approach effectively tackled two persistent issues: prompt sensitivity and hallucination. By implementing standardizing structured input prompts and ontological constraints, we created a consistent framework that reduced output variability and uncontrolled generation, enhancing both reliability and maintainability in production environments.

The culmination of our research is reflected in the training dynamics, which revealed distinct patterns between adaptation strategies. Full block adaptation demonstrated stable convergence and superior final performance, while attention-only adaptation showed

higher training volatility. Notably, even when comparing full block adaptation at rank 4 with attention-only adaptation at rank 32—which has approximately three times the parameter capacity—the full block approach demonstrated more stable and comparable performance. This finding underscores that thoughtful architectural choices can outweigh raw parameter capacity in achieving stable and effective model adaptation. Both approaches benefited from higher rank configurations, though with diminishing returns. Most significantly, the system maintained robust performance across different element types, excelling in entity extraction, binary and higher-order relation identification with their attributes, validating our fundamental approach to knowledge extraction and structuring.

7.2 Limitations

Despite the promising outcomes, this research is not without its limitations. First, working with LLMs poses significant computational challenges, particularly when scaling to larger architectures or conducting iterative experiments. While our LoRA adaptation effectively reduced the parameter space, the base model's inference still demands substantial computational resources.

Our second limitation stems from the reliance on synthetic training data. Data quality inherently depends on the template-generating LLMs' performance, and synthetic data may not fully capture real-world text complexities. Additionally, the necessary manual validation step creates a bottleneck in template generation. It's worth noting that our evaluation results, while remarkably high, should be interpreted as a proof of concept. These scores are partially attributable to using the same methodological approach for both training and testing datasets, even though we changed the variations. This suggests the need for future validation with more diverse evaluation methods.

7.3 Future Work

The results presented in this thesis demonstrate the significant potential of Large Language Models in the domain of automated knowledge extraction, particularly in transforming unstructured text into ontology-guided knowledge representations. While our dual-task framework and adaptation strategies have shown promising results across various domains, several promising research directions emerge that could substantially enhance the system's capabilities and broaden its applications.

One potential direction for future research lies in the integration of multi-modal capabilities. While our current framework effectively handles textual data, extending it to process and extract knowledge from diverse data types—including images, videos, and audio, could significantly enhance its utility. This extension would require developing unified representation schemes capable of capturing cross-modal relationships while preserving ontological consistency. The challenge extends beyond mere multi-modal processing; it necessitates the development of robust validation frameworks that can effectively verify

the accuracy and coherence of extracted knowledge across these diverse modalities. Such an expansion would enable more comprehensive knowledge extraction from rich, multimodal data sources that are increasingly common in modern information systems.

Another potential direction involvesexpanding the system's multilingual capabilities. This expansion should investigate the development of language-agnostic ontological frameworks that can effectively capture knowledge across different linguistic structures. Additionally, research into zero-shot and few-shot transfer learning techniques could prove particularly valuable for low-resource languages, potentially enabling knowledge extraction in contexts where training data is scarce.

Finally, a particularly promising direction involves leveraging knowledge distillation techniques to bridge the gap between large, highly capable language models and more efficient, specialized models for knowledge extraction. This approach would involve using state-of-the-art large language models as "teachers" to transfer their sophisticated understanding of knowledge structures and relationships to smaller, more deployable "student" models. The knowledge distillation process could focus specifically on transferring capabilities relevant to structured knowledge extraction, potentially allowing smaller models to achieve performance levels approaching those of much larger models while maintaining computational efficiency.

Bibliography

[1] Jay Alammar. *The Illustrated Transformer*. (Accessed on 06/06/2024). Jan. 2024. URL: https://jalammar.github.io/illustrated-transformer.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].

[3] Prabin Bhandari. *A Survey on Prompting Techniques in LLMs*. 2024. arXiv: 2312.03740 [cs.CL].

[4] Paul F. Christiano et al. "Deep reinforcement learning from human preferences". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 4302–4310. ISBN: 9781510860964.

[5] E. F. Codd. "A relational model of data for large shared data banks". In: *Commun. ACM* 13.6 (June 1970), pp. 377–387. ISSN: 0001-0782. DOI: 10.1145/362384.362685. URL: https://doi.org/10.1145/362384.362685.

[6] Contributors to Wikimedia projects. *Natural language processing - Wikipedia*. (Accessed on 30/05/2024). May 2024. URL: https://en.wikipedia.org/w/index.php?title=Natural_language_processing&oldid=1222328215.

[7] DeepLearning.AI. *Natural Language Processing (NLP) - A Complete Guide*. [Online; accessed 30. May 2024]. Jan. 2023. URL: https://www.deeplearning.ai/resources/natural-language-processing.

[8] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: 10.18653/V1/N19-1423. URL: https://doi.org/10.18653/v1/n19-1423.

[9] Abhimanyu Dubey et al. *The Llama 3 Herd of Models*. 2024. arXiv: 2407.21783 [cs.AI]. URL: https://arxiv.org/abs/2407.21783.

[10] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[11] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[12] Jiwoo Hong, Noah Lee, and James Thorne. *ORPO: Monolithic Preference Optimization without Reference Model*. 2024. arXiv: 2403.07691 [cs.CL]. URL: https://arxiv.org/abs/2403.07691.

[13] Neil Houlsby et al. *Parameter-Efficient Transfer Learning for NLP*. 2019. arXiv: 1902.00751 [cs.LG]. URL: https://arxiv.org/abs/1902.00751.

[14] Jeremy Howard and Sebastian Ruder. *Universal Language Model Fine-tuning for Text Classification*. 2018. arXiv: 1801.06146.

[15] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL]. URL: https://arxiv.org/abs/2106.09685.

[16] Lei Huang et al. *A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions*. 2023. arXiv: 2311.05232 [cs.CL].

[17] Pere-Lluıs Huguet Cabot and Roberto Navigli. "REBEL: Relation Extraction By End-to-end Language generation". In: *Findings of the Association for Computational Linguistics: EMNLP 2021*. Ed. by Marie-Francine Moens et al. Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 2370–2381. DOI: 10.18653/v1/2021.findings-emnlp.204. URL: https://aclanthology.org/2021.findings-emnlp.204.

[18] Albert Q. Jiang et al. *Mistral 7B*. 2023. arXiv: 2310.06825 [cs.CL]. URL: https://arxiv.org/abs/2310.06825.

[19] Albert Q. Jiang et al. *Mixtral of Experts*. 2024. arXiv: 2401.04088 [cs.LG]. URL: https://arxiv.org/abs/2401.04088.

[20] Prashant Johri et al. "Natural Language Processing: History, Evolution, Application, and Future Work". In: *Proceedings of 3rd International Conference on Computing Informatics and Networks*. Singapore: Springer, Mar. 2021, pp. 365–375. ISBN: 978-981-15-9712-1. DOI: 10.1007/978-981-15-9712-1_31.

[21] Mahboob Alam Khalid, Valentin Jijkoun, and Maarten de Rijke. "The Impact of Named Entity Normalization on Information Retrieval for Question Answering". In: *Advances in Information Retrieval*. Ed. by Craig Macdonald et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 705–710. ISBN: 978-3-540-78646-7.

[22] Alessandro Lamberti. "Deep Learning Model Optimization Methods". In: *Neptune* (May 2024). [Online; accessed 9. Jun. 2024]. URL: https://neptune.ai/blog/deep-learning-model-optimization-methods.

[23] Mike Lewis et al. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension*. 2019. arXiv: 1910.13461 [cs.CL]. URL: https://arxiv.org/abs/1910.13461.

[24] Xiang Lisa Li and Percy Liang. *Prefix-Tuning: Optimizing Continuous Prompts for Generation*. 2021. arXiv: 2101.00190 [cs.CL]. URL: https://arxiv.org/abs/2101.00190.

[25] Igor Melnyk, Pierre Dognin, and Payel Das. *Knowledge Graph Generation From Text*. 2022. arXiv: 2211.10511 [cs.CL]. URL: https://arxiv.org/abs/2211.10511.

[26] Dat Quoc Nguyen and Karin Verspoor. “End-to-End Neural Relation Extraction Using Deep Biaffine Attention”. In: *Advances in Information Retrieval*. Ed. by Leif Azzopardi et al. Cham: Springer International Publishing, 2019, pp. 729–738. ISBN: 978-3-030-15712-8.

[27] OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774.

[28] Long Ouyang et al. *Training language models to follow instructions with human feedback*. 2022. arXiv: 2203.02155.

[29] Aravindpai Pai. “What is Tokenization in NLP? Here’s All You Need To Know”. In: *Analytics Vidhya* (June 2024). [Online; accessed 03. June 2024]. URL: https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp.

[30] Balkrishna Pandey. “How Tokenization and Encoding Work in LLM?” In: *Goglides Dev xn–uh8h* (Dec. 2023). [Online; accessed 03. June 2024]. URL: https://www.goglides.dev/bkpandey/how-tokenization-and-encoding-work-in-llm-4cih#what-is-encoding.

[31] Ebrahim Pichka. “What are Query, Key, and Value in the Transformer Architecture and Why Are They Used?” In: *Medium* (Oct. 2023). [Online; accessed 6. Jun. 2024]. URL: https://towardsdatascience.com/what-are-query-key-and-value-in-the-transformer-architecture-and-why-are-they-used-acbe73f731f2.

[32] Alec Radford and Karthik Narasimhan. “Improving Language Understanding by Generative Pre-Training”. In: 2018. URL: https://api.semanticscholar.org/CorpusID:49313245.

[33] Alec Radford et al. “Language Models are Unsupervised Multitask Learners”. In: (2018). URL: https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf.

[34] Rafael Rafailov et al. *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*. 2023. arXiv: 2305.18290 [cs.LG]. URL: https://arxiv.org/abs/2305.18290.

[35] John W. Ratcliff and John A. Obershelp. *Ratcliff/Obershelp pattern recognition*. https://www.nist.gov/dads/HTML/ratcliffObershelp.html. July 1988.

[36] David E. Rumelhart and James L. McClelland. “Learning Internal Representations by Error Propagation”. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1987, pp. 318–362.

[37] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG]. URL: https://arxiv.org/abs/1707.06347.

[38] Melanie Sclar et al. “Quantifying Language Models’ Sensitivity to Spurious Features in Prompt Design or: How I learned to start worrying about prompt formatting”. In: *arXiv* (Oct. 2023). DOI: 10.48550/arXiv.2310.11324. eprint: 2310.11324.

[39] Rico Sennrich, Barry Haddow, and Alexandra Birch. *Neural Machine Translation of Rare Words with Subword Units*. 2016. arXiv: 1508.07909 [cs.CL].

[40] Wei Shen, Jianyong Wang, and Jiawei Han. "Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions". In: *IEEE Transactions on Knowledge and Data Engineering* 27.2 (2015), pp. 443–460. DOI: 10.1109/TKDE.2014.2327028.

[41] Amit Singhal. *Introducing the Knowledge Graph: things, not strings*. https://blog.google/products/search/introducing-knowledge-graph-things-not/. (Accessed on 06/30/2024).

[42] Rohan Taori et al. *Stanford Alpaca: An Instruction-following LLaMA model*. https://github.com/tatsu-lab/stanford_alpaca. 2023.

[43] Gemma Team et al. *Gemma 2: Improving Open Language Models at a Practical Size*. 2024. arXiv: 2408.00118 [cs.CL]. URL: https://arxiv.org/abs/2408.00118.

[44] Gemma Team et al. *Gemma: Open Models Based on Gemini Research and Technology*. 2024. arXiv: 2403.08295 [cs.CL]. URL: https://arxiv.org/abs/2403.08295.

[45] Mistral AI team. *Cheaper, Better, Faster, Stronger | Mistral AI | Frontier AI in your hands*. https://mistral.ai/news/mixtral-8x22b/. (Accessed on 08/24/2024).

[46] Mistral AI team. *Large Enough | Mistral AI | Frontier AI in your hands*. https://mistral.ai/news/mistral-large-2407/. (Accessed on 08/25/2024).

[47] Mistral AI team. *Mistral NeMo | Mistral AI | Frontier AI in your hands*. https://mistral.ai/news/mistral-nemo/. (Accessed on 08/25/2024).

[48] Python Development Team. *difflib — Helpers for computing deltas — Python 3.13.0 documentation*. https://docs.python.org/3/library/difflib.html. (Accessed on 10/07/2024).

[49] Alex Thomas. *Natural Language Processing with Spark NLP*. Sebastopol, CA, USA: O'Reilly Media, Inc. ISBN: 978-1-49204776-6. URL: https://www.oreilly.com/library/view/natural-language-processing/9781492047759/ch01.html.

[50] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL]. URL: https://arxiv.org/abs/2307.09288.

[51] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: 2302.13971 [cs.CL].

[52] Lewis Tunstall, Leandro von Werra, and Thomas Wolf. *Natural Language Processing with Transformers, Revised Edition*. Sebastopol, CA, USA: O'Reilly Media, Inc., May 2022. ISBN: 978-1-09813679-6. URL: https://www.oreilly.com/library/view/natural-language-processing/9781098136789.

[53] Sowmya Vajjala et al. *Practical Natural Language Processing*. Sebastopol, CA, USA: O'Reilly Media, Inc., June 2020. ISBN: 978-1-49205405-4. URL: https://www.oreilly.com/library/view/practical-natural-language/9781492054047.

[54] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[55] Jonathan J. Webster and Chunyu Kit. “Tokenization as the initial phase in NLP”. In: *Proceedings of the 14th Conference on Computational Linguistics - Volume 4*. COLING ’92. Nantes, France: Association for Computational Linguistics, 1992, pp. 1106–1110. DOI: 10.3115/992424.992434. URL: https://doi.org/10.3115/992424.992434.

[56] Jason Wei et al. *Finetuned Language Models Are Zero-Shot Learners*. 2022. arXiv: 2109.01652.

[57] Yonghui Wu et al. *Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. 2016. arXiv: 1609.08144 [cs.CL].

[58] Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. *BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models*. 2022. arXiv: 2106.10199 [cs.LG]. URL: https://arxiv.org/abs/2106.10199.

[59] Wenxuan Zhou et al. *UniversalNER: Targeted Distillation from Large Language Models for Open Named Entity Recognition*. 2024. arXiv: 2308.03279 [cs.CL]. URL: https://arxiv.org/abs/2308.03279.

8 Appendix

This appendix demonstrates the output of our knowledge extraction system through representative examples. Each example represents a different domain, showing the complete extraction pipeline: from source text to ontological structure to hypergraph representation.

8.1 Recruitment Example

Text

On 2024-08-05, HealthPlus Global, a Healthcare Technology company based in Toronto, Canada, posted a job opening for a Senior Data Scientist position. The role, reporting to Chief Data Officer, requires 5+ years of experience in Machine Learning, Python, and TensorFlow. The position offers a competitive salary range of $120,000 - $160,000 with additional benefits including comprehensive health benefits, stock purchase plan, and flexible working hours.
Carlos Ramirez, a Talent Acquisition Lead at HealthPlus Global, is spearheading the recruitment process. On 2024-08-10, Emma Svensson, currently a Data Analyst at DataWorks Inc., submitted an application through Glassdoor. Emma Svensson's resume highlighted 6 years of relevant experience, proficiency in Machine Learning, Python, and R, as well as a Master's in Data Science from University of Toronto.
The initial screening process began on 2024-08-12 when Jasmine Lee, the HR Specialist, conducted a preliminary review of Emma Svensson's application. Based on the positive screening results, Carlos Ramirez invited Emma Svensson for a phone interview with Dr. Rajiv Menon, the Lead Data Scientist, scheduled for 2024-08-15.
During the 45-minute phone interview, Dr. Rajiv Menon assessed Emma Svensson's technical knowledge, problem-solving skills, and cultural fit. The candidate performed well, demonstrating strong expertise in advanced machine learning models and data visualization expertise. As the next step, Emma Svensson was asked to complete a data analysis project focusing on predictive modeling and data cleaning and preprocessing.
Emma Svensson submitted the assessment on 2024-08-20 and achieved an impressive score of 90%. The assessment results were reviewed by Linda Thompson, the Senior Data Analyst, who provided positive feedback on the candidate's accurate predictions and clear and insightful visualizations.
Impressed by the candidate's performance, Carlos Ramirez coordinated an onsite interview for 2024-08-25. The interview panel consisted of:

1. Michael Chen (Data Engineering Manager) - Technical interview
2. Sara Ibrahim (Product Manager) - System design discussion
3. Lucas Martins (Business Analyst) - Behavioral interview
4. Yuki Sato (UX Researcher) - Team fit assessment

The onsite interview lasted for 4 hours, during which Emma Svensson participated in various technical discussions, whiteboard coding sessions, and behavioral scenarios. Post-interview feedback was collected using the company's Applicant Tracking System (ATS).
On 2024-08-28, the hiring committee, led by Dr. Helen Zhao, the Chief Data Officer, convened to review the interview feedback and make a decision. The committee noted Emma Svensson's strengths in cross-functional teamwork and innovative problem-solving, as well as their potential areas for growth in big data technologies and real-time data processing.
After careful consideration and a unanimous vote, the hiring committee decided to extend an offer to Emma Svensson. Carlos Ramirez prepared the offer package, which was reviewed and approved by Thomas Nguyen, the HR Director, on 2024-08-30.
On 2024-09-01, Carlos Ramirez extended a formal job offer to Emma Svensson with the following details:

- **Position**: Senior Data Scientist
- **Starting salary**: $150,000 per year
- **Sign-on bonus**: $10,000
- **Stock options**: 20,000 shares vesting over 4 years
- **Additional benefits**: Comprehensive health benefits, stock purchase plan, flexible working hours, and tuition reimbursement
- **Proposed start date**: 2024-10-01

Emma Svensson received the offer and requested a few days to review the terms. After some consideration and a brief negotiation regarding remote work flexibility, Emma Svensson accepted the offer on 2024-09-05. The final terms included a slight adjustment to allow 3 remote work days per week, which was approved by Dr. Helen Zhao.
With the offer accepted, Sophie Martin, the HR Assistant, initiated the onboarding process. This included:

1. Background check and reference verification (completed on 2024-09-10)
2. Preparation of new hire paperwork
3. Coordination with IT for equipment setup
4. Assigning a mentor, David Lee (Senior Data Scientist), for the first 90 days

Emma Svensson is scheduled to start on 2024-10-01, with a comprehensive orientation program planned for the first week. Isabella Rossi, the Data Science Team Lead, has prepared a 30-60-90 day plan to ensure a smooth integration into the team and project ramp-up.
This successful recruitment process demonstrates HealthPlus Global's commitment to attracting top talent and maintaining a rigorous, fair, and efficient hiring practice.

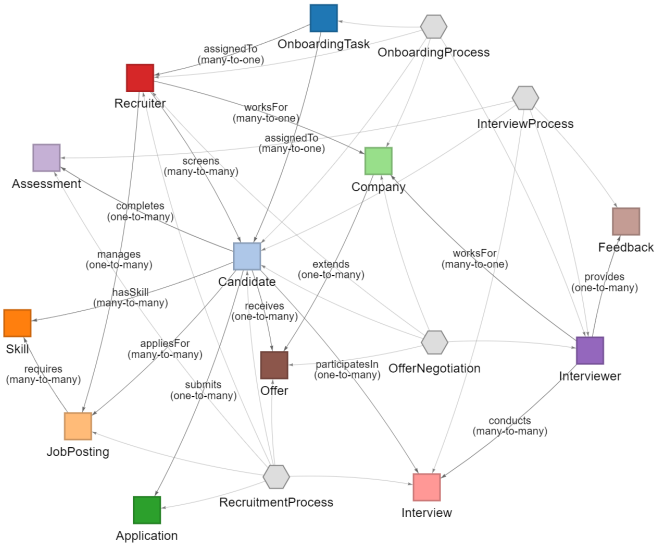


Figure 8.1: Ontology output

Figure 8.1 represents the data model as an ontology where rectangles define entity types: purple for interviewers, red for recruiters, blue for onboarding tasks, orange for skills, green for company entities, light blue for candidates, light blown for feedback and dark brown for offers. Gray hexagons represent process types that act as complex relationships (hyperedges), and connecting lines specify allowed relationships and their cardinalities between entities.

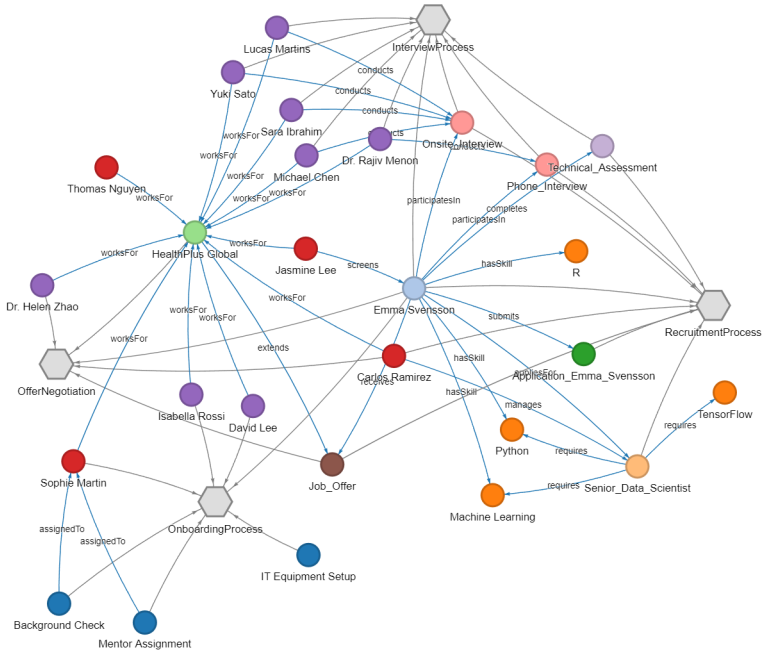


Figure 8.2: Hypergraph output

Figure 8.2 implements this ontology as a hypergraph where colored circles (nodes) represent instances of the entity types matching their corresponding colors from the ontology. The gray hexagonal hyperedges connect these nodes according to the relationship rules defined in the ontology, capturing the actual recruitment, interviewing, and onboarding processes at HealthPlus Global.

8.2 Healthcare Example

Text

Dr. Amara Okafor, ein/e Onkologie am Sunshine Medical Center in Lagos, behandelt Chioma Adebayo wegen Stadium II Brustkrebs. Der Behandlungsplan umfasst eine Kombination aus Chirurgie, Chemotherapie mit Paclitaxel und Strahlentherapie. Dr. Amara Okafor verschrieb Paclitaxel in einer Anfangsdosis von 175 mg/m² alle alle drei Wochen für 4 Zyklen, beginnend am 15. März 2024. Chioma Adebayo erlebte moderat Übelkeit als Nebenwirkung, die mit Ondansetron behandelt wurde. Nach dem zweiten Zyklus ergaben Blutuntersuchungen einen besorgniserregenden Rückgang der weißen Blutkörperchen, was Dr. Amara Okafor veranlasste, die Dosis von Paclitaxel um 20% für die verbleibenden Zyklen zu reduzieren. Chioma Adebayos Tumorgröße verringerte sich um 40% nach Abschluss der Chemotherapie. Sie unterzog sich dann einer Lumpektomie, die von Dr. Olayinka Mensah am 10. Juli 2024 durchgeführt wurde, gefolgt von 6 Wochen Strahlentherapie, beginnend am 1. August 2024, verabreicht von der Strahlentherapeutin Dr. Aisha Diallo.

Figure 8.3 represents the medical treatment ontology where rectangles define entity types: orange for doctors (Arzt), green for medications (Medikament), blue for patients (Patient), purple for treatments (Behandlung), light blue for diseases (Krankheit), red for tests (Test), pink for medical centers (Zentrum), and peach for surgical procedures (Chirurgie). Gray hexagons represent hyperedges types like diagnosis (Diagnoseprozess), treatment planning (Behandlungsplan), and side effect management (Nebenwirkungsmanagement). The connecting lines specify allowed relationships and their cardinalities between entities in German, such as "behandeltVon" (treated by) or "unterziehtSich" (undergoes).

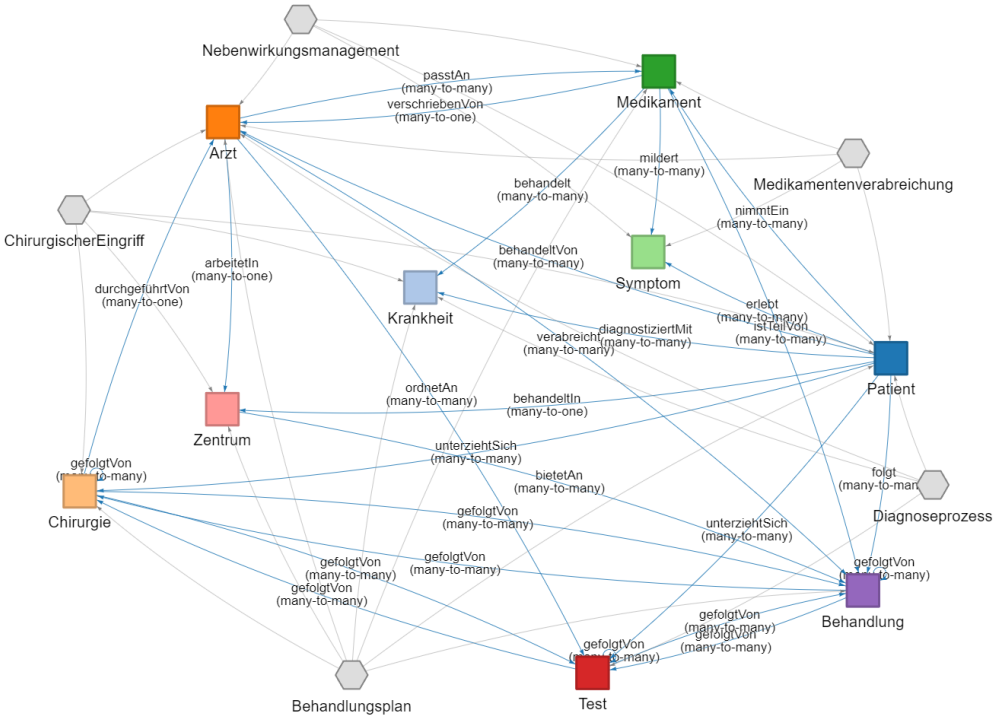


Figure 8.3: Ontology output

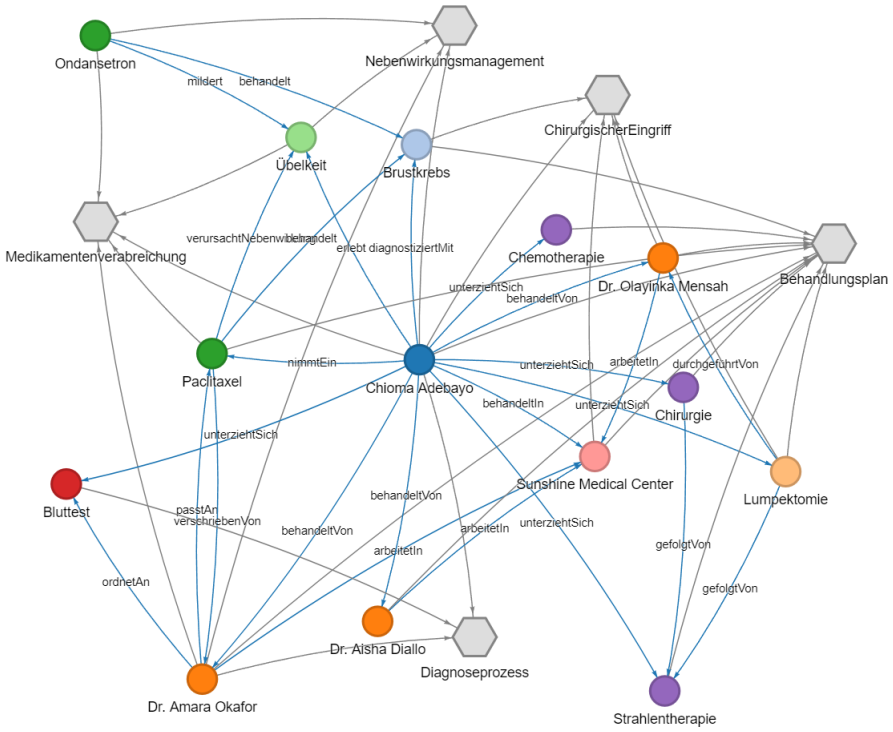


Figure 8.4: Hypergraph output

Figure 8.4 implements the ontology as a hypergraph showing a breast cancer (Brustkrebs) treatment case. Colored circles (nodes) represent instances matching their ontology types: orange for doctors (Dr. Olayinka Mensah, Dr. Aisha Diallo, Dr. Amara Okafor), green for medications (Ondansetron, Paclitaxel), blue for the patient (Chioma Adebayo), purple for treatments (Chemotherapie, Strahlentherapie, Chirurgie), and pink for the medical facility (Sunshine Medical Center). The gray hexagonal hyperedges connect these nodes according to the relationship rules defined in the ontology, capturing the actual diagnosis, treatment planning, and side effect management processes.

8.3 Education Example

Text

Marcus Rodriguez, a Business Administration major at Bay Area College located in San Francisco, has enrolled in 3 courses for the Spring 2025 semester. The courses include International Business taught by Dr. Jennifer Lee, Business Analytics taught by Dr. Thomas Wilson, and Strategic Management taught by Dr. Amanda Patel. Marcus Rodriguez is on Excellent academic standing and has a GPA of 3.95. The student has registered for Entrepreneurship Society as part of their extracurricular commitments. Marcus Rodriguez is receiving financial aid from Federal Grant amounting to $12000 per semester. The student's advisor, Prof. Lisa Martinez, reviews their academic progress every Bi-Monthly. As of 2024-11-15, Marcus Rodriguez has completed 108 credits and is on track to graduate in 2025. Marcus Rodriguez has participated in President's List and has received the Diversity in Business Scholarship scholarship. Marcus is working on improving his public speaking skills through the Communication Skills Workshop.

Figure 8.5 represents the academic management ontology where rectangles define entity types: orange for universities, brown for financial aid, purple for courses, pink for instructors, blue for advisors, light blue for academic progress, green for academic issues and extracurricular activities, peach for academic standing, and pink for academic honors. Gray hexagons represent process types such as enrollment, scholarship awards, academic support, and extracurricular participation. The connecting lines specify allowed relationships and their cardinalities between entities, such as "enrolledIn" (many-to-many), "advisedBy" (many-to-one), and "hasAcademicProgress" (one-to-one).

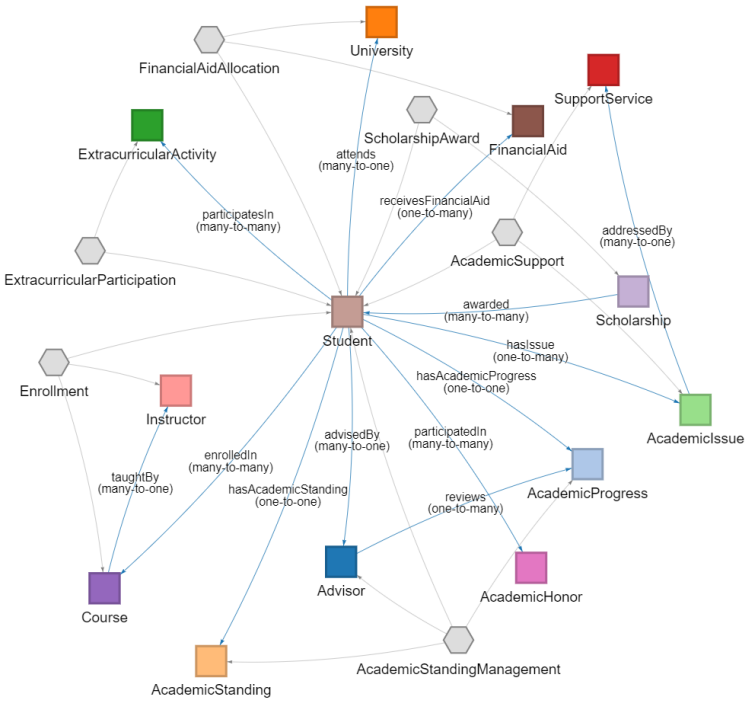


Figure 8.5: Ontology output

Figure 8.6: Hypergraph output

Figure 8.6 present the hypergraph based on the ontology, showing a student's (Marcus Rodriguez) academic journey. Colored circles (nodes) represent instances matching their ontology types: orange for Bay Area College, pink for instructors (Dr. Jennifer Lee, Dr. Amanda Patel, Dr. Thomas Wilson), blue for advisor (Prof. Lisa Martinez), purple for courses (Strategic Management, International Business, Business Analytics), brown for financial aid (Federal Grant), and peach for academic standing (Excellent). The gray hexagonal hyperedges connect these nodes according to the relationship rules defined in the ontology, capturing the actual enrollment, academic support, and extracurricular participation processes, including Marcus's participation in the Entrepreneurship Society and achievement of the President's List honor.

8.4 Finance Example

Text

Global Financial Bank
123 Finance Street, Moneyville, MV 12345
Contact: +1 (555) 123-4567 | Website: `www.globalfinancialbank.com`

Account Statement

Statement Period: 2024-02-01 to 2024-02-29

Account Information

Account Holder: John Doe
Account Number: 4872-9135-6028
Account Type: Checking
Branch: Downtown Branch (DTN001)

Account Summary

Opening Balance: $5,000.00
Total Credits: $3,000.00
Total Debits: $2,500.00
Closing Balance: $5,500.00

Transaction Details

Date	Description	Reference	Debit	Credit	Balance
2024-02-05	Payroll Deposit	DEP-12345	—	$2,000.00	$7,000.00
2024-02-10	Online Purchase - Retailer XYZ	POS-67890	$1,500.00	—	$5,500.00
2024-02-20	ATM Withdrawal	ATM-24680	$1,000.00	—	$4,500.00

Additional Account Information

Available Credit: $10,000.00
Credit Limit: $10,000.00
Interest Rate: 2.5% APR
Next Statement Date: 2024-03-31

Account Analytics

Average Daily Balance: $5,250.00
Largest Credit: $2,000.00 on 2024-02-15
Largest Debit: $1,500.00 on 2024-02-10
Number of Transactions: 3

Important Notices

1. Our mobile banking app will undergo maintenance on March 15, 2024.
2. New savings account options available! Visit our website for more information.

Reward Points Summary

Opening Balance: 1,000
Points Earned: 150
Points Redeemed: 50
Closing Balance: 1,100

Contact Information

Customer Service: +1 (555) 987-6543
Lost/Stolen Card: +1 (555) 111-2222
Online Banking: `www.globalfinancialbank.com/online`

Please review this statement carefully. Report any discrepancies within 60 days of the statement date. Unreported errors will be deemed as accepted.

Generated on: 2024-03-01 at 23:59:59
Statement Reference: STMT-2024-02-JD

Member FDIC. Equal Housing Lender.

Figure 8.7 represents the banking system ontology where rectangles define entity types: green for banks, light blue for account holders, green for accounts, orange for transactions, peach for notices, blue for statements, pink for branches, and red for reward programs. Gray hexagons represent hyperedge types such as account activity, account analytics, reward points activity, and statement summary. The connecting lines specify relationships between entities, such as "containsTransaction", "holdsAccount", "generateStatement", and "containsNotice".

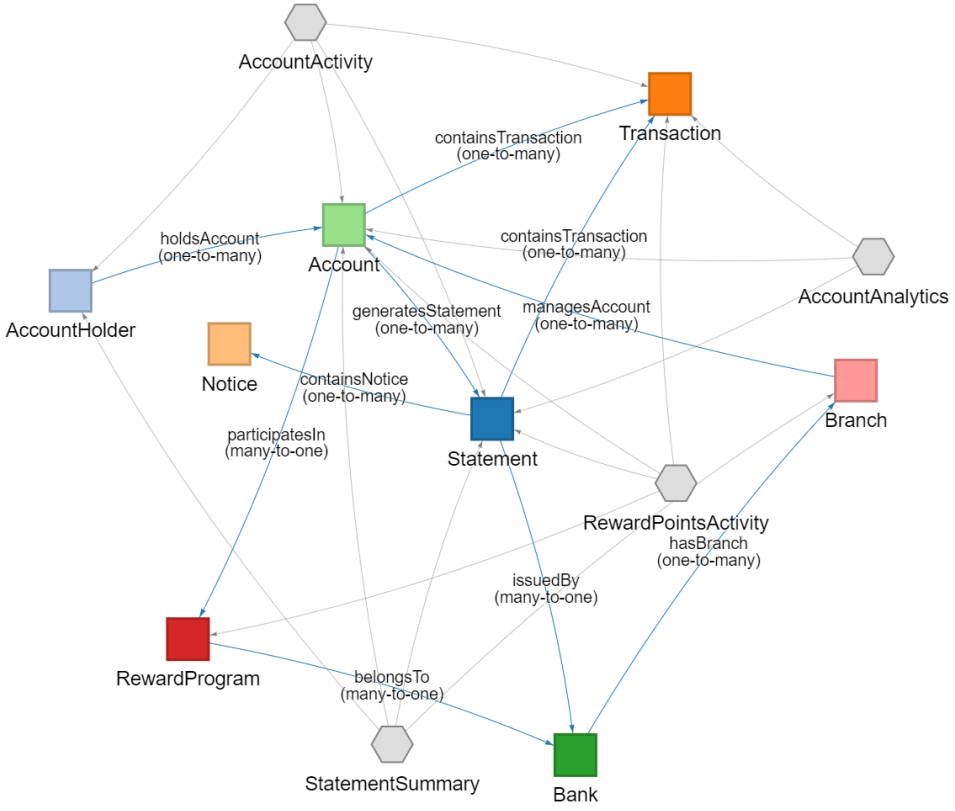


Figure 8.7: Ontology output

Figure 8.8 present the hypergraph showing a specific account activity instance. Colored circles (nodes) represent instances matching their ontology types: green for Global Financial Bank and checking account (Account_Checking_4872-9135-6028), blue for statement record (Statement_4872-9135-6028_Stmt-2024-02-jd), orange for transactions, pink for Downtown Branch, light blue for account holder (John Doe), and peach for account notices. The gray hexagonal hyperedges connect these nodes according to the relationship rules defined in the ontology, capturing the actual account transactions, statement generation, and reward program participation processes for a specific customer's account.

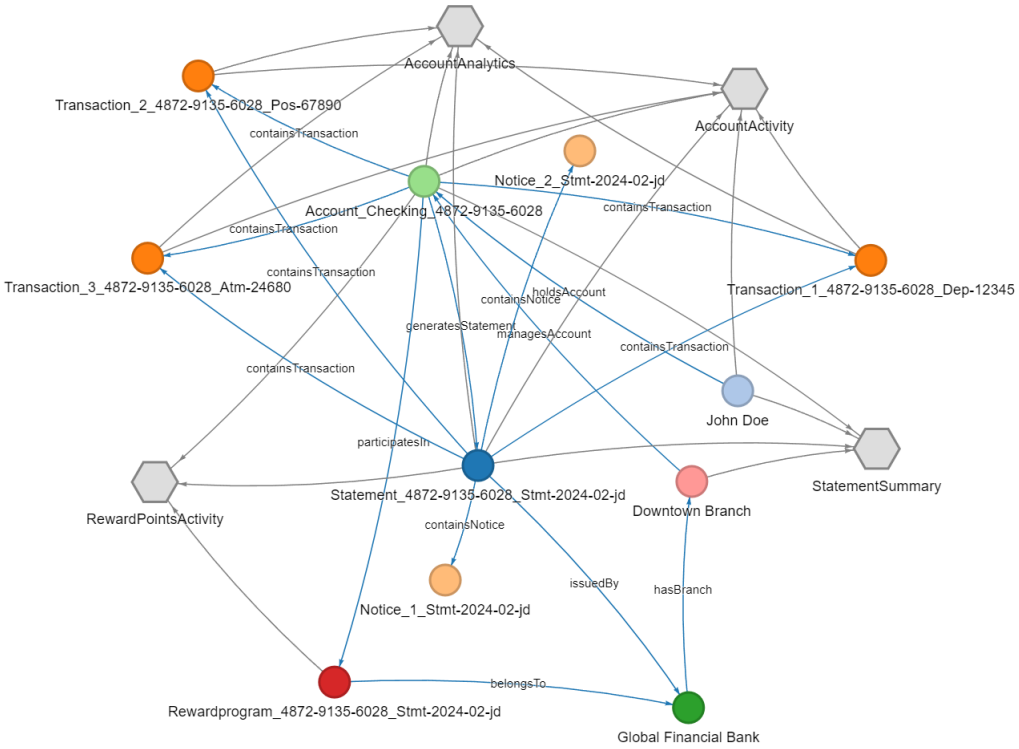


Figure 8.8: Hypergraph output