TU WIEN Informatics

Contextual Rule Learning for Knowledge Graphs Using Large Language Models

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering and Internet Computing

by

Orwa Suman, BSc.
Registration Number 01429050

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr. Katja Hose
Assistance: Univ.Prof. Dr. Emanuel Sallinger

Vienna, February 11, 2025

Orwa Suman	Katja Hose

Erklärung zur Verfassung der Arbeit

Orwa Suman, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 11. Februar 2025

Orwa Suman

Acknowledgements

I would like to express my deepest gratitude and respect to my supervisors, Univ.Prof. Dr. Katja Hose and Univ.Prof. Dr. Emanuel Sallinger, for their invaluable guidance, patience, and unwavering support throughout this thesis. Their profound insights and immense knowledge have shaped this research and inspired me to continue the research journey in this field, and I am truly honored to have learned under their guidance.

A heartful appreciation goes to my family for their steadfast encouragement, consistent support, and belief in my abilities. In particular, I am deeply grateful to my father, whose presence since the very beginning of this journey has been a constant source of strength and reassurance, and his invaluable insights have contributed to this work. Furthermore, his dedication to reviewing this thesis and his unwavering belief in me remain unforgettable, and for that, I am profoundly thankful. Also, I sincerely thank my dear mother, whose unconditional love and kind words have given me the confidence to persevere through challenges and complete this work.

Finally, thanks to my friends for their consistent help and motivating words throughout this thesis.

Abstract

Knowledge Graphs are crucial in developing knowledge-based AI systems. They infer knowledge using logical rules that describe the data in the knowledge graph. Rule miners aim to extract rules from knowledge bases using various techniques, primarily relying on the KB structured information to find co-occurrences and correlations between facts. Most rule miners, such as AMIE, cannot utilize the context of the facts and incorporate domain knowledge when mining rules, and they also face challenges with high resource utilization, leading to missed correlations and suboptimal rule discovery. This thesis explores the integration of Large Language Models (LLMs) into the rule mining process of AMIE to enhance rule quality, contextual awareness, and computational efficiency.

The approach systematically integrates LLM usage to realize LLM-based rule initialization, evaluation, and refinement using Google Gemini. The integration balances rule quality and quantity improvements with computational feasibility. Extensive experiments and evaluations are conducted to assess the impact of the LLM on rule mining, specifically on the accuracy, completeness, and efficiency. Additionally, the experiments are designed to measure the impact of the LLM optimization and the prompt. The results show that incorporating LLMs leads to more rules generated, improving overall coverage of the rule mining process. However, LLM optimization and prompt clearness play a crucial role in the rule quality and computational efficiency.

Contents

CHAPTER 1

Introduction

Artificial intelligence (AI) is the field that aims to simulate human intelligence to enable machines to perform complex tasks that require reasoning and connected knowledge. It has emerged in the last decade as the fastest-growing field in computer science, led by unprecedented advancements in machine learning, deep learning, and natural language processing. In the recent few years, the field of generative AI has experienced a massive boost with the release of OpenAI's Large Language Model (LLM), GPT-3. This LLM was a groundbreaking achievement that allowed public users to access artificial intelligence services for the first time, enabling them to perform many tasks that were impossible before. Additionally, the public use of ChatGPT allowed researchers and developers to improve the LLM by collecting various data, which helped release many subsequent versions of ChatGPT. The breakthrough of Generative AI also started a competition between tech-leading companies such as Google, OpenAI, Microsoft, and others to provide AI services to public users, which was an important factor in growing AI research. Now, many LLMs compete to provide the best quality AI services, such as ChatGPT from OpenAI [Ope25], Gemini from Google [Goo25], Claude from Anthropic [Ant25], and many more.

One important subfield of AI systems is knowledge-based systems that manage structured data to derive knowledge and find hidden correlations in the data. Normally, knowledge-based systems rely on a Knowledge Graph (KG) that facilitates different machine learning techniques to derive structured knowledge that is managed using graph principles. Knowledge in knowledge-based systems is structured and based on a knowledge base (KB) that contains entities and relationships that are considered facts. New knowledge in knowledge-based systems can be derived using rules that describe the knowledge base. Such systems are normally domain-specific, i.e., most suitable for use in a specific domain, and use cases such as information retrieval, recommendation systems, and chatbots.

Rule mining is an important technique in data mining and knowledge-based systems that is concerned with finding rules that best describe the facts in a knowledge base.

These rules are then used to find correlations, hidden patterns, and relationships between entities, in addition to identifying errors and deviations in the knowledge base. Many rule miners use various techniques to extract rules from knowledge bases, such as Association Rule Mining under Incomplete Evidence (AMIE) [GTHS13] [GTHS15] [LGS20], Sherlock [SDEW10], Rudik [SDEW10], and Evoda [WSS+22]. Most rule miners do not facilitate the contextual information in KB facts or incorporate domain knowledge to find new rules with new KB relations, thereby missing valuable information that could improve the quality of the mined rules and the mining process. This work proposes using LLM in rule mining to improve the mining result and address rule miners' contextual information and domain knowledge shortage, focusing on exploring the integration between LLM and the AMIE rule miner.

1.1 Background and Motivation

AMIE[GTHS13] is a seminal rule miner that extracts rules from knowledge bases given in turtle format, containing triples in RDF style that represent facts in the form of subject-predicate-object [BP25]. Each fact in the KB consists of two entities connected with a relation. For example, the KB fact $motherOf(Bob, Alice)$ corresponds to the turtle format representation $< Alice, motherOf, Bob >$, which specifies that Alice is the mother of Bob. AMIE mines horn rules from KBs, i.e., logical rules consisting of a conjunction of facts that imply an outcome fact. These rules describe the facts in the KB. For example, the logical rule

$$fatherOf(X, Y) \wedge marriedTo(Y, Z) \implies motherOf(Z, X)$$

describes the parental relationships: if the father Y of a person X is married to a person Z, then person Z is the mother Z of person X. AMIE operates under the Partial Closed World Assumption (PCA) that states if an object is known to be *true* for a subject in a KB, then all objects for this subject-predicate pair are assumed to be known. In other words, under PCA, we can assume that if a fact exists for an entity in the KB, the absence of similar facts for the same entity implies they are *false*. For example, if the KB contains the fact *Emily lives in Paris*, the absence of the information *Emily lives in London* means that AMIE does not live in London, but would treat the absence of the information *Bob lives in London* as *unknown*. This allows AMIE to find rules in incomplete KBs by computing meaningful confidence scores, which allows mining rules that are not overgeneralized or undergeneralized. In contrast to PCA, the Open World Assumption (OWA) assumes missing facts as unknown, which impedes finding definitive conclusions because the absence of missing facts does not imply falsehood.

Additionally, AMIE, in its latest version, AMIE3.5 [LGS20], supports multi-threading and uses an in-memory database to boost the performance when mining rules on large KBs. AMIE algorithm to mine rules can be divided into three main phases: (1) initialization phase, which initializes an initial queue of rules containing simple implication rules of all

KB relations; (2) evaluation phase to decide if a rule can be outputted, and (3) refinement phase in which new rules are built by enumerating all possible KB relations, and add these rules to the queue to be processed in later iterations of the mining process.

Despite its great performance, AMIE faces challenges such as high resource utilization due to the use of an in-memory database that can use up to 500 GB of memory, as well as high CPU usage when using multithreading. Additionally, like other rule miners, AMIE lacks the ability to use the context of each fact to build rules, which is valuable information that can improve the mining process in various phases. Additionally, if a rule does not have enough evidence in the KB that passes a given threshold, AMIE throws it away, although the rule context is valid, i.e., the relations contexts are relevant to each other, and the logical connection between their variables is valid.

Motivated by the limitations of AMIE and the shortage of contextual evaluation of KB facts in most rule miners, this work aims to explore the possibilities of integrating an LLM into the AMIE algorithm to enhance its rule-mining capabilities. It aims at exploring which phases of the mining process can be improved. The LLM's ability to generate contextually relevant text and retrieve domain-specific knowledge makes it a valuable complement to traditional rule miners. More specifically, the LLM can generate rules by incorporating domain knowledge of the facts given as input and reasoning with their context to find logical correlations among them. Additionally, the LLM can support AMIE in evaluating the context of rules, which includes checking if the logical connection between the facts in a rule is valid. Finally, using the LLM could improve the runtime by eliminating the need for exhaustive mining operations that AMIE performs. Chapter 4 explains in detail the mining process of AMIE.

1.2 Research Questions

This work is to improve AMIE's mining process by integrating LLM-based tasks across multiple phases of the algorithm. Specifically, the LLM will complement AMIE in the initialization phase and support it in evaluating and discovering new rules. It will also explore how LLM capabilities can be integrated into AMIE and what improvements to the mining result the LLM can bring. Additionally, it performs exhaustive experiments and detailed analysis of the integration result to measure the quality of the LLM-based tasks.

The research questions of this work are as follows.

- How can LLMs be integrated into AMIE to improve the efficiency of rule mining? Which metrics can be used to measure the results?

  AMIE is a complex approach that was developed in 2013 and is continuously optimized. This thesis explores the phases of the mining algorithm and which code parts can be modified to integrate LLM. Additionally, it studies how to measure the quality of the rules affected by the LLM-based tasks.

- Which LLM approaches can be combined and used in different phases of the rule mining process?

  LLM approaches can perform similar tasks in different phases of the mining process. For instance, they can generate rules that are parsed and used in AMIE to complement the initialization phase. The LLM can also support the refinement phase of AMIE by generating rules to be parsed and used in AMIE. Therefore, the two similar approaches can be combined and used in both phases.

- Which steps of the mining process can be adjusted to minimize calls to LLM while maintaining good results, low runtime, and scalability, thereby balancing the LLM benefits and drawbacks?

  LLM use can be expensive in runtime and cost, as it requires time to process prompts and generate responses, especially when exhaustively making requests and parsing responses, such as evaluating the context of every rule in the queue. The runtime of AMIE can be affected, which leads to the question of whether LLM use makes sense if it affects the runtime to the degree that it makes no sense to implement its integration in AMIE. Therefore, this work explores steps that can lead to a balanced trade-off between good results of LLM integration and overall process performance.

1.3 Approach

This work aims to design independent procedures to perform different operations using the LLM, which are called LLM tasks. These tasks would consume inputs from the AMIE algorithm and define the output. Therefore, multiple steps are defined to integrate LLM within AMIE. The LLM used is Google Gemini. This section shows the steps to integrate Google Gemini into AMIE.

1.3.1 Code Analysis and Establishing LLM Communication

The first step is to understand the AMIE code and identify which code parts correspond to which phases of the mining algorithm. The LLM, i.e., Google Gemini, also provides APIs for third-party applications to send requests containing the prompt and parameters used to control the content generation. Therefore, it is essential to implement a Gemini client class responsible for communication with Gemini APIs, i.e., sending prompts and receiving responses. This work uses the Vertex AI service that Google Cloud Platform (GCP) provides, enabling communication between Java applications and Google Gemini.

1.3.2 Design LLM Tasks

In the phases of AMIE where the LLM can be integrated, prompt templates are created that provide clear instructions and relevant information to the LLM to support the mining process. Additionally, controlled content generation is used through Gemini to

ensure consistency in responses and avoid unstructured free-text generation, with Vertex AI enforcing structured outputs using JSON schemas. Each prompt or request to Gemini follows a defined JSON schema, ensuring standardized and reliable responses.

With the foundational setup introduced, the LLM tasks can be implemented. Each LLM task requires input data from AMIE, which would be included in the corresponding prompt. Additionally, within each task, the LLM response in the form of JSON string is parsed. Flow tracking and statistic gathering are designed to measure the quality of each LLM task, ensuring an objective evaluation of the results.

As the LLM tasks depend on AMIE to provide inputs, and the output of each LLM task would be used in the AMIE mining process, the LLM tasks are integrated within the AMIE algorithm within the corresponding phases identified previously, resulting in an algorithm where all LLM tasks work together alongside with AMIE algorithm.

1.3.3 Experimental Evaluation

The integration of LLM tasks within AMIE is implemented to enable single LLM tasks when running AMIE, i.e., the algorithm's execution is controlled by a set of control parameters that enable individual LLM tasks. Furthermore, the LLM tasks are completely separated from AMIE, allowing for the highest degree of flexibility in integrating LLM within AMIE.

To evaluate the effectiveness of each LLM task and its integration into AMIE, comprehensive experiments are conducted. The evaluation of the results aims to measure the improvements made by each LLM task to the mining result of AMIE, the impact on the runtime, and the ability of the LLM to discover new relations that consent with the KB.

1.4 Main Contribution

This work explores the efficiency of Large Language Models in rule mining. Specifically, this thesis investigates the integration of LLMs into AMIE to enhance the quality, contextual relevance, and efficiency of mined rules. The primary contributions of this research are summarized as follows:

1. **Identification of Integration Points for LLMs in Rule Mining**
    - A systematic analysis of the AMIE algorithm to determine the phases where LLMs can provide meaningful enhancements.
    - Definition of LLM-based tasks complementing AMIE in rule generation and evaluation.
2. **Development of LLM-Assisted Rule Mining Mechanisms**
    - Design of LLM tasks tailored to improve AMIE's rule discovery process

- Development of LLM integration within AMIE without disrupting its core functionalities.

3. **Empirical Evaluation and Performance Analysis**
   - Execution of comprehensive experiments to evaluate the effectiveness of LLM-enhanced rule mining on real-world knowledge bases.
   - Quantitative and qualitative analysis of rule quality, mining efficiency, and computational performance.
4. **Optimization Strategies for Balancing LLM Utility and Computational Costs**
   - Investigation of strategies to mitigate the trade-offs between LLM-enhanced contextual reasoning and computational overhead.
   - Implementation of optimized workflows that reduce LLM invocation frequency while preserving the quality of the mining process.

1.5 Structure of the Thesis

This thesis is structured into nine chapters, beginning with an Introduction chapter that outlines the background and motivation, the research questions, and the approach. The second chapter covers the theoretical background of this work, including fundamental concepts such as knowledge graphs, rule mining, and generative AI. Chapter Three overviews existing rule-mining techniques, including related work to similar approaches, followed by Chapter Four, which describes AMIE in detail. Chapter Five describes the approach of integrating LLMs into AMIE, detailing different LLM-based tasks, whereas the implementation of the approach is described in Chapter Six. The experiments and the analysis of the results are discussed in Chapters Seven and Eight, respectively. Finally, this thesis concludes in Chapter Nine.

CHAPTER 2

Theoretical Background

Knowledge engineering is an essential part of data science and is concerned with building knowledge-based systems, a branch of artificial intelligence systems. This section explains the theoretical background of knowledge engineering and the key concepts of knowledge-based systems and outlines the differences between them and artificial intelligence systems. Additionally, this section describes Generative AI techniques and Large Language Models (LLMs), how they are built, and what they can do.

2.1 Information Systems

In computer science, many terms in the area of information technology are widely used, and some are misused in the wrong contexts. The following describes the different terms used in the area of information technology.

2.1.1 Knowledge and Intelligence

The terms data, information, and knowledge are widely used in computer science. The term understanding is also helpful in understanding bigger concepts such as intelligence. Data is the smallest part of any information system. It is raw symbols in any form that have no meaning of themselves, as they simply exist. Information is processed data that has a meaning, typically with a relational connection to other data. Information can answer simple questions with short answers, i.e., giving answers about something like "who", "what", "where" and "when" questions. Information is found widely in relational databases, where raw data are connected to derive answers to simple questions [BCM04].

Knowledge is a collection of different information with a focus on answering bigger questions, in particular "how" questions, that provide longer answers with a sequence of information that contains a pattern. Knowledge is meant to be useful, but it does not provide an integration to infer further knowledge as itself. For example, answering simple

multiplication questions (i.e., $2 * 2 = 4$) requires knowledge, but this knowledge cannot be used to answer more complex multiplications without a systematic analysis, which is considered a higher level of knowledge called understanding. The term "understanding" refers to a cognitive and analytical process that processes existing knowledge to derive new knowledge. A comparison between understanding and knowledge can be projected on the difference between learning and memorizing [BCM04].

Intelligence is the ability to understand knowledge and apply it to solve problems. Human intelligence encompasses multiple dimensions that include analytical, creative, practical, and emotional intelligence. In computer science, artificial intelligence simulates human intelligence but is limited to fewer dimensions [gen23].

2.1.2 Knowledge-Based vs Artificial Intelligence Systems

Artificial intelligence systems aim to simulate human intelligence using various techniques to enable machines to perform tasks that require reasoning. These systems typically require stored knowledge to learn, operate, and solve problems where the data is not necessarily structured. Famous sub-fields of AI systems are machine learning, deep learning, language models, and natural language processing [gen23]. AI systems normally imply the use of data to train a model, which understands the relationships between data and the patterns of information using special algorithms. This results in knowledge that is implied within the model itself and, in most cases, cannot be inspected.

Knowledge-based systems are a subset of AI systems, i.e., all knowledge-based systems are AI systems. They facilitate knowledge management, which is concerned with deriving knowledge from data, storing and processing it, and using it in various applications. Artificial intelligence systems are a broader concept that includes deriving the knowledge to train models and predicting new knowledge.

In contrast to AI systems, knowledge-based systems rely heavily on structured data to derive knowledge. They perform reasoning by applying logical rules to the data, where the knowledge they obtain comes completely from the data contained in Knowledge Bases. A Knowledge-based system is normally used in a specific domain, where its predictions are considered reliable. The subsequent sections explain knowledge-based systems, how they derive knowledge, how knowledge is represented, and finally, the concept of logical rules.

2.1.3 Knowledge Bases and Knowledge Graphs

A Knowledge Base (KB) is a collection of structured and unstructured information. It stores data in a document-like format, such as simple text files, turtle format, and other simple representations (see Section 2.1.4). The main idea of knowledge bases is to statically store organized information for fast information retrieval, i.e., answering specific queries. Knowledge bases normally contain billions of facts about the world, as they contain information such as Vienna is the capital of Austria.

A Knowledge Graph (KG) is a structured representation of information with a focus on modeling the relationships (graph edges) between entities (graph nodes). They use graph structures and graph theory principles to store, create, evolve, and maintain knowledge. In contrast to knowledge bases, knowledge graphs aim at finding relationships, semantics, and patterns between entities and are often dynamic, allowing for inferring new knowledge under the concept of graph evolution. Knowledge graph schemas are normally designed to be flexible and to handle complex relationships between entities, as well as allowing for easy maintenance and extendibility [Wis25]. Knowledge graphs are based on a data model to represent the knowledge, whereas the data model choice for knowledge representation depends on the domain and the application of the knowledge graph. Section 2.1.4 describes different data models to represent the knowledge.

A knowledge-based system relies on a knowledge graph, as in a knowledge graph, knowledge is represented in various formats and is inferred using various techniques. Section 2.1.4 will explain how knowledge is represented and organized in knowledge graphs.

2.1.4 Knowledge Representation Frameworks

As mentioned earlier, knowledge graphs make use of graph structure and graph theory to handle knowledge. Knowledge bases, however, do not require any graph structures, but they still make use of different data representation formats such as ontological representation using turtle format, which stores facts as triples, or database format.

Ontological Representation

An ontology is a domain-specific, formal representation of knowledge. It defines concepts (classes), their properties, and the relationships between them. It is normally used to enable semantic interoperability by providing a common understanding of a domain, which is what knowledge graphs build on. Ontologies are a powerful representation of structured data, as they can be extended easily, allowing for high flexibility and interoperability. Additionally, they can be designed using different ontology design patterns (ODPs) specifically tailored to a specific domain.

Ontologies are represented using a graph structure, where the nodes represent the concepts, entities, or classes, while the edges represent the relationships between the classes, i.e., the object and data properties. Encoding an ontology is often realized using a formal language such as Web Ontology Language (OWL), Resource Description Language (RDF), or its extension RDF Schema (RDFS).

RDF. The Resource Description Framework is a general-purpose language for representing information in the web [BP25]. It provides a data model for fast integration of data, which can be used to represent graph structures, having graph nodes represented by RDF individuals and graph edges represented by RDF triples. RDF stores the entire graph in one table called the RDF graph, and it is an object-oriented data model comprising entities represented by unique resource identifiers (URIs). These entities, referred to as resources, represent the graph nodes in a graphical representation of RDF graphs.

Relationships, however, are expressed as edges. A relationship source is called the subject, a relationship destination is called the object, and the relationship itself is a predicate property. This representation of a graph containing resources and relationships is the Subject (S) - Predicate (P) - Object (O) form, referred to as RDF triples. RDF triple syntax is as $< Subject, Predicate, Object >$, and it provide a deeper understanding of data by combining full-text search, graph analytics, and logical reasoning [DMM00]. An example of an RDF triple is the information $< Alice, motherOf, Bob >$, which corresponds to the fact "*Alice is the mother of Bob*".

RDF can be serialized in different formats such as Turtle, JSON-LD, and XML. Turtle format is widely used as it allows an RDF graph to be completely written in a compact and natural text form [BP25]. Querying knowledge in RDF triple store formats is often realized using SPARQL querying language, which offers a wide range of support for graph-like queries such as multi-hop and recursive queries.

Despite their benefits, RDF triple stores are not suitable for the efficient storage of large graph data. The triple store format has a high overhead in storage, in addition to performance drawbacks due to inefficient queries requiring computationally intensive navigation through the graph nodes and edges traversal [DMM00].

Relational Databases and NoSQL Database

Another option to represent knowledge is to use typical relational databases. Here, raw data is stored in tables, and knowledge is represented based on relations between the tables. Although graph structures are not directly omitted in relational database management systems (RDBMS), they can be realized through foreign key relations, leading to an indirect representation of knowledge. This vague representation of graph structures, however, struggles with realizing complex relationships between entities, which is the core idea of knowledge graphs. Complex queries and recursive operations are inefficient in SQL, as relational databases are optimized for relational joins, not graph traversals. Additionally, the rigid schema of RDBMS makes them less flexible for knowledge graph evolution. Therefore, relational databases are not necessarily the first choice for modelling a knowledge graph.

In contrast to relational databases, NoSQL databases are better data models for knowledge graphs, as they offer a flexible schema, built-in graph operations, and high scalability. Additionally, they support core knowledge graph queries such as multi-hop queries and recursive operations. NoSQL databases offer a wide range of options, as this concept includes any non-RDBMS database, including graph databases and document databases. Graph databases are normally the ideal data model of choice for representing the knowledge in a knowledge graph.

Graph Databases

A database that uses graph structures to represent and persist data is called a graph database. A property graph model is a model used in graph database design. It contains

connected entities, represented as nodes, and relationships between entities, represented as edges. Additionally, nodes and edges are labelled to indicate their role in the application domain. A relationship between two entities is an edge between two nodes. Relationships are directed edges, they donate semantics and assign a start node and an end node. Nodes and edges also have properties, whereas in edges properties are often quantitative, such as weight, cost, distance, rating, or time interval. Finally, nodes and edges are defined by a unique identifier [Pok15].

In the context of databases, graph data can be generally represented by tables in a regular Relational Database Management System (RDBMS), and using SQL or Datalog, graph queries can be realized. Alternatively, graphs in graph databases can also be represented in other structures, such as RDF, JSON, or XML formats.

In contrast to regular relational databases, indexes are used to speed up access to data by creating an additional data structure. In graph databases, however, this approach may lead to inefficiencies, particularly for operations that involve traversing many edges between nodes. To process data effectively, graph databases use a property called index-free adjacency, where every node in the graph contains an index of other nearby nodes it connects to. This approach allows each node to directly hold references to its adjacent nodes without the need for an index lookup, thereby enhancing the execution time of queries that involve navigating through multiple nodes [Pok15].

There are many examples of graph databases, such as Neo4j, ArangoDB, JanusGraph, and Amazon Neptune.

2.1.5 Data Quality

Data quality measures the degree to which the data in a dataset meet certain criteria, i.e., the so-called data quality dimensions, which are specific characteristics. The following define the most relevant data quality dimensions inside a knowledge base [SSPA+12].

- **Accuracy (Correctness):** degree to which data is correct, i.e., corresponds to real-world values, as well as reliable and certified.
- **Completeness:** degree to which a KB contains all necessary, meaningful, and relevant data representing the real-world system or entity, with no missing or null values.
- **Consistency:** violation of semantic constraints defined for a KB.
- **Timeliness:** to which age the data is appropriate to be used, i.e., how long does the data describe real-world facts without causing a delay
- **Validity:** information that conforms to given rules and formats, i.e., birth dates conforming to a specific date format.

The data quality includes many more definitions, such as relevance, interpretability, etc. These dimensions are essential for a knowledge-based system, affecting its effectiveness, reliability, and trustworthiness. Therefore, many approaches have been introduced to ensure data quality, such as automated validation techniques, data cleansing and preprocessing, and continuous monitoring and updates. Automated validation techniques use software tools to verify the data against predefined rules without human intervention, which is the opposite of the latter two approaches to ensure data quality, which involve manual human inspection of the data against existing criteria to ensure data quality [Cha24].

Automated validation techniques are preferable when designing knowledge-based systems, as they allow for validating large amounts of data quickly, allowing for uniform validation of data, in addition to the option of real-time validation. One famous technique for validating knowledge bases in RDF format is the use of Shapes Constraint Language (SHACL), which allows for defining constraints that allow for defining shapes to describe how RDF data should be structured. These shapes include specifying RDF graph nodes (entities), which should conform to the defined constraint of the shape. SHACL can validate RDF graph structure, i.e., data cardinality such as minimum and maximum cardinality, RDF graph nodes such as validating data types and formats, and the logical connection between graph nodes.

This work focuses on completeness and correctness. Ensuring the correctness of facts in knowledge bases, which generally contain billions of facts about the world, can be challenging. KBs can contain false data, missing information, and inconsistent schema definitions. Additionally, there is no way to know if a KB is complete, i.e., considering a proportion of the facts are correct, we cannot know what proportion of facts in the real world they cover [GRAS17]. That's why many rule miners make assumptions about the KB, such as the Partial Closed-World Assumption, as explained in Section 2.2.2.

2.2 Key Concepts

In information systems, a traditional database uses a database management system (DBMS) to organize and manage data access. Database theory focuses on querying the data and its description. Deductive databases, however, facilitate logical reasoning to organize facts and use logical rules to infer new facts and relationships from existing facts, i.e., deduce new facts using logical programming languages such as Datalog and Prolog. A logical rule is a set of logical clauses with logical connections. This section explains the key concepts of deductive databases and logic programming.

2.2.1 Deductive Databases

Logic programming is a declarative programming paradigm based on predicate logic where programs are written as a set of logical statements that include declaring facts, rules, and queries. A *declarative* programming paradigm specifies what information

should be retrieved rather than how to retrieve this information. It is the opposite of *imperative* programming paradigm that implements clear steps to retrieve information. Facts in a logical program are considered known truths, rules are the logical relationships between facts, and queries represent the questions to be answered with logical reasoning. In a logical program, *constants* are objects such as people, places, food, etc. Facts in logical programs are expressed as *atoms*, which are expressions formed from n-ary relationships. For example, the atom `livesIn(Bob, Vienna)` expresses the fact "*Bob lives in Vienna*", having *livesIn* the relation (binary relation), constant subject *Bob*, and constant object *Vienna*. To represent relationships between objects without explicitly declaring constants, *variables* can be used to achieve variability of atoms. This means an *atom* can have variables at the subject and/or object positions.

A *rule* is an expression consisting of atoms having (shared) variables to express logical relationships among them. It contains a *body* and a *head*, which can be a conjunction of atoms. In a rule, the body implies the head, i.e., the truth logical evaluation of the body implies the truth logical evaluation of the head. There are several types of rules, such as horn rules and associative rules, explained in detail in Section 2.3. Having the base concepts covered, we can define a **deductive database** as "a finite collection of facts and rules" [Gen05].

Datalog

Querying deductive databases requires using a querying language such as Datalog or Prolog, which allows for writing logical programs consisting of rules. This work focuses on Datalog and uses its notation to express facts, rules, and queries.

Table 2.1 describes the Datalog notation and syntax. Note that in the context of this work, $\boldsymbol{B}$ and H are the body and the head of the rule, respectively, which are vectors of atoms.

Expression	**Definition**	**Notation**
term	constants and Variables	a, ..., z; A, ..., X
atom	facts describing a n-ary relationship between two terms	$Relation(t_1, t_2, ..., t_n)$
rule	logical connection between atoms	**H <= B**
program	a finite set of rules	P
query	a program and a relation	*(P, Relation)*

Table 2.1: Datalog syntax

As mentioned in Section 2.1.4, RDF triples represent facts using the form Subject (S) - Predicate (P) - Object (O) in knowledge bases. The predicate is an equivalent term to relation, and facts are expressed as atoms. Therefore, we can express facts in a knowledge base as relation(s, o), or in short *r(s, o)*. Subject (s) and Object (o) are always constants in knowledge bases, but when using atoms in a rule, subject and objects can also be (shared) variables between atoms.

Knowledge Bases Characteristics

As knowledge bases consist of atoms, we can define characteristics that describe different aspects of the facts in a knowledge base.

Functionality. The functionality is a characteristic of a KB relation that measures the number of unique values for a given subject. A relation can occur multiple times in the KB; therefore, it can be functional or non-functional, depending on whether it assigns a unique object for each subject. For example, the relation `hasBirthdate` is known to have only one unique object (the birth date) for each subject (e.g., person, living being, etc.), while a subject (parent) for the relation `hasChild` can be associated with multiple objects (children). The former relation example is a functional relation, and the latter is a non-functional relation. Formally, the functionality *fun* of a relation r is defined as:

$$fun(r) = \frac{\#x : \exists y \; r(x, y)}{\#(x, y) : r(x, y)}$$

where $\#\alpha$ denotes the number of α that fulfil a condition. In other words, the functionality of a relation r is the proportion of how many objects a relation associates a subject with to the number of relations having the same subject.

An additional definition for the functionality of a relation r is "quasi-functional", where the functionality value *fun(r)* is close to 1. An example of this is the relation `isCitizenOf`, where a subject (person) can be associated with a few objects (country). The more objects a relation has for a subject, the closer the functionality of this relation to 0 [GRAS17].

Completeness. As mentioned in Section 2.1.5, completeness refers to the KB characteristic that measures to which extent a KB covers the facts of the real world. Galárraga et al. [GRAS17] define completeness of a KB κ via a hypothetical ideal KB κ^*. A KB κ is complete for a query q if it delivers the same result as κ^* running the same query q. This means that the pair of a subject s and relation r is complete in a KB if the objects delivered by κ are a subset of the objects the pair delivers on κ^*. For example, the relation *hasCapital* and subject *South Africa* is complete on KB κ if it contains all three capitals of South Africa. Completeness is an important characteristic of a KB in the context of Rule Mining, as explained in Section 2.3.

Cardinality. A relation in a KB has a cardinality value, defined as the minimum and maximum number of subjects and objects a relation can be associated with. For example, for the relation *hasParent*, each subject can have exactly two objects, i.e., the father and mother. The relation *hasAuthor* can have exactly one object for each subject, etc. [Mun25]. The cardinality of relations in the KB can be easily validated and declared using constraint languages such as SHACL, as described in Section 2.1.5. Similarly, the cardinality of the subject-object pair s, r, is defined as the number of unique objects o the subject s can be associated with for a relation r. This is a crucial characteristic in a KB used in rule quality metrics, as explained in Section 2.3.

2.2.2 Completeness Concepts

The definition of completeness of KBs can be extended to the completeness of a knowledge-based system, which is an essential property to measure the comprehensiveness of such systems. Knowledge-based systems operate under three main completeness concepts.

Closed World Assumption

The closed world assumption (CWA) states that if a fact is unknown, it is assumed to be *false*, i.e., does not hold in the real world. Under this assumption, a KB is considered complete. For example, if there is no fact in the KB to state that person X is the president of country Y, then this missing fact is *false*, i.e., person X is not the president of country Y. In general, this assumption holds for most *person* entities in a KB, as only one person is the president of the country Y [GRAS17].

This assumption is used in databases, where a piece of missing information (a record) is assumed to be *false*. Similarly, a logic program assumes all facts not derived from the rules to be *false*. For example, if person X is not listed in the employees' table of a database, then this person is assumed not to be an employee.

Open World Assumption

The open world assumption (OWA) states that a missing fact of a KB can be *true* in the real world, and it is treated as unknown rather than *false*. This assumption is applied in the semantic web, where a missing piece of information is treated as unknown, and the data in the KB is treated as incomplete [GRAS17]. Additionally, Knowledge Graphs operate under this assumption to evolve knowledge.

Most artificial intelligence systems, including knowledge-based systems, assume that the data in a KB are incomplete because it allows for a flexible treatment of the data and evolving knowledge. Using the example from before and assuming the database operates under OWA, if person X is not listed in the employee's table, then this person's employment status is assumed unknown, i.e., the person could be an employee, but we lack the information to be sure. Still, operating under OWA makes the reasoning harder, as the missing facts do not imply falsehood, leading to no definitive conclusion. To mitigate the limitations associated with the Open World Assumption (OWA) and Closed World Assumption (CWA), many knowledge-based systems consider the implications of the Partial Closed World Assumption.

Partial Closed World Assumption

In contrast to OWA and CWA, the partial closed world assumption (PCA) assumes that some parts of the KB are complete and others are not. This is a mix between OWA and CWA, and it is less strict than CWA in assuming completeness by assuming some unknown facts as false and some as false. Specifically, a subject-relation pair is assumed complete if at least one object exists for this pair. Hence, PCA assumes completeness

only for entities with an object for a relationship while not making any assumptions about other unexisting objects for the entity associated with the subject. For example, if person X's nationality does not exist in the KB, then PCA would not assume the person has no nationality, while CWA would make such an assumption [GRAS17].

PCA is most suitable for Knowledge Graphs, as it treats some information as *true* and other missing information as *unknown* while not making wrong assumptions on missing data like CWA, and at the same time, making definitive conclusions, unlike OWA. This allows a Knowledge Graph to evolve and derive new knowledge easily.

2.3 Rule Mining

Data mining refers to the concept of extracting unknown information from data in a knowledge base. The information extracted includes hidden knowledge, logical rules, constraints, and data patterns. Rule mining is a data mining technique that extracts rules from large datasets by identifying associations, correlations, or dependencies between data attributes. The extracted rules are used to predict new knowledge in a domain-specific knowledge-based application.

In the context of this thesis, two main types of rules are relevant:

- Association rules: express relationships between items in a dataset and follow the format:
- Horn Rules: logical implications expressed in the form of Horn Clauses, commonly used in logic programming and knowledge representation.

This work focuses on mining horn rules, which are logic-based rules that predict one fact. Association rules are widely used in data mining and pattern discovery, while Horn rules are essential in logical reasoning, AI, and knowledge representation.

2.3.1 Association Rule Mining

Association rules are implications in the form $X \implies Y$. These rules express relationships between items in a dataset, where an item can be anything depending on the data. They are often used in market-based analysis to describe customer habits. In an association rule, X and Y are sets of items called the antecedent and the consequent, respectively [ZB03]. For example, the rule $\{Burger, Fries\} \Rightarrow \{Soda, Ketchup\}$ specifies that when a customer buys the items *Burger* and *Fries*, then he will buy the items *Soda* and *Ketchup* as well, i.e., the items in the antecedent is associated with the items in the consequent. Association rule mining is a rule mining technique that aims to mine association rules from large KBs and is used to discover interesting relationships between items.

In general, mining association rules from KBs involves two main steps: (1) items that usually appear together in the KB are identified, and (2) a rule generation algorithm is

applied to extract rules from the KB, such as Apriori, FP-Growth, and ECLAT [ZB03]. During the rule generation step, evaluation measures such as support and confidence are used to assess the quality of each rule. If a rule does not meet the predefined support or confidence thresholds, it is discarded by most association rule mining algorithms. Otherwise, it is included in the final set of discovered rules, providing valuable insights into the relationships between items in the KB.

2.3.2 Horn Rule Mining

Now, we explain in detail the theory behind horn rules, which are the type of rules this work focuses on.

Preliminaries

The concepts from Section 2.2.1 will be used in this work from now on. The term *atom* will be used to represent the facts of a knowledge base. A *rule* is a logical statement that allows inference of new knowledge. It consists of a head atom, which is the inferred fact(s), and a body consisting of a conjunction of *atoms*. Each atom in a rule can have variables at the subject and/or object positions.

First-Order logic, also called *Predicate logic*, is a formal system used in artificial intelligence to express statements about subjects, relationships, and objects with quantifiers (exists, all), variables, and logical connection [Bar77]. A *Horn rule* is a special case of first-order logic expressed as an implication, having a single atom in the head. A *Horn rule's body* consists of a conjunction of *atoms*, meaning that all atoms in the body are connected using the logical AND ($\wedge$) operation in the form $B_1 \wedge B_2 \wedge ... \wedge B_n$. For example, the rule

$$motherOf(X, Y) \wedge fatherOf(X, Z) \implies marriedTo(Y, Z)$$

is a horn rule, having the atoms *motherOf(X, Y)* and *fatherOf(X, Y)* in the rule body, and the atom *marriedTo(Y, Z)* in the rule head. This example illustrates the logical connection between the three atoms using shared variables, which means that the atom in the head can only be positive if both atoms in the body are positive. Of course, there can be exceptions to this rule, i.e., having the mother and father of a person not married, but for most cases, this rule would hold. This applies to any rule in the real world [GTHS13].

A rule *instantiation* is the substitution of all rule variables by entities. Variables at the subject and object positions in an atom can be substituted by constants to express facts. Substituting the variables in an atom results in an atom instantiation. A rule *prediction* is the instantiated head atom of a rule, having all atoms of the rule body also instantiated, and the resulting facts exist in the KB. For example, the instantiation of the above rule can be

$$motherOf(Max, Lisa) \wedge fatherOf(Max, Tony) \implies marriedTo(Lisa, Tony)$$

2.3.3 Horn Rule Properties

One of the biggest challenges in rule mining is to extract meaningful rules that best describe the facts in the KB. The quality of the rule set mined by a rule mining algorithm depends on certain measures defined by the mining algorithm. Generally, rules in a knowledge-based system have properties associated with the KB that can be used by mining algorithms to assess the quality of each rule and, hence, the quality of the mined rule set. The properties are specific to each rule type, but as this work focuses on mining horn rules, this section describes the properties of a horn rule, focusing on AMIE's definitions of these properties.

Support. A rule prediction is the number of rule instantiations in the KB. In other words, the number of *true* predictions a rule makes in the KB [GTHS13] [LGS20]. Formally, the support of a rule R mined from a KB K is defined as

$$support(R) =\mid \{p \mid (K \wedge R \models p) \wedge p \in K\} \mid$$

The support is a useful property of a rule used in most rule mining algorithms. More specifically, support is used by AMIE as one of the quality metrics, as explained in Section 4.2.3.

Head-Coverage. By definition, support is an absolute number. The rule's proportional variant of the support is called the *head-coverage*, which is the ratio of head atom instantiations that are correctly predicted by the rule, i.e., the number of *true* predictions made by the rule on the KB to the number of head atom instantiations found in the KB. The *head-coverage* specifies how many known facts are covered by the rule. Formally, the head-coverage is defined as [LGS20]

$$hc(B \Rightarrow r(x,y)) = \frac{support(B \Rightarrow r(x,y))}{|(x,y) : r(x,y) \in K|}$$

Standard Confidence. To measure the quality of a rule, it is important to consider not only the *true* predictions it makes but also the *false* predictions. Therefore, similar to the *head-coverage*, the ratio of *true* predictions of the *true* predictions and *false* predictions is the *standard confidence* of a rule. It specifies how often a rule makes correct predictions. Formally defined for a rule R as follows:

$$conf\left(\overrightarrow{B} \Rightarrow r(x,y)\right) := \frac{support\left(\overrightarrow{B} \Rightarrow r(x,y)\right)}{\#(x,y) : \exists z_1, \ldots, z_m : \overrightarrow{B}}$$

where $R = (B \Rightarrow r(x,y))$. The confidence is considered the accuracy of a rule, and the support is considered the relevance of a rule [LGS20].

Rules mined by AMIE should pass certain confidence and support thresholds to be considered in the output set of rules, which is considered part of AMIE evaluation, as shown in Chapter 4.

PCA Confidence. The standard confidence of a rule operates under the closed-world assumption (CWA) because a prediction made by the rule that does not exist in the KB, i.e., a missing prediction, is considered a *false* prediction. When operating under the partial closed-world assumption (PCA), it is crucial to correctly determine when to call a missing prediction a false prediction. The *PCA confidence* of a rule specifies that if at least one object o exists for a subject s and relation r, i.e., the atom r(s, o) exists in the KB, then we assume we know all objects of this subject-relation pair. In other words, PCA-confidence estimates rule reliability without assuming missing facts are false. If a rule predicts a fact that is not present in the KB, and at least one other fact is present in the KB (an object associated with a subject for the relation), then this prediction is considered false. Formally, the PCA confidence for a rule R is defined as

$$pca - conf(B \Rightarrow r(x,y)) = \frac{support(B \Rightarrow r(x,y))}{|\{(x,y) \mid \exists y' : B \wedge r(x,y')\}|}$$

where $R = (B \Rightarrow r(x,y))$ [GTHS15].

Connected Rule. Two atoms in a rule are connected if they share arguments, whether a variable or an entity, i.e., a variable or an entity, is found in both atoms. A rule is *connected* if every atom is connected to every other atom of the rule, including transitive connections. For example, the rule "$motherOf(X,Y) \wedge fatherOf(X,Z) \implies marriedTo(Y,Z)$" is connected because the atoms "$motherOf(X,Y)$" and "$fatherOf(X,Z)$" are connected through variable X, and they are both connected with the atom "$marriedTo(Y,Z)$" through the transitive variables $X \rightarrow Y, Y \rightarrow Z$.

Closed Rule. A rule is *closed* if every variable in the rule appears at least twice. The rule from above is closed because each variable appears exactly twice in the rule.

2.3.4 Prediction Model

Logical rules extracted from a KB using a rule miner are generally used to predict new knowledge, find errors in the data, and more. With the KB, rules make a prediction model to predict outcomes based on the inputs, similar to an AI system's traditional machine learning model. Such models are called rule-based predictive models [HMR21]. More specifically, rules can be used differently in domain-specific applications. Rule predictions can be used as a basis for further computations to find accurate outcomes or are considered absolute facts, depending on the application.

Machine Learning Models vs. Rule-Bases Models

Machine Learning is a sub-field of study in artificial intelligence that applies statistical algorithms and models that can learn patterns and correlations from the data to predict

new outcomes based on inputs. Like rule-based predictive models, they aim to perform specific tasks without explicitly being programmed [Mah20], but they differ in how they learn and make predictions. For instance, machine learning models are generally considered black-box, as it is difficult to interpret how a prediction was made, while rule-based models are highly transparent, making each prediction easily traceable. Machine learning still advances in other areas, such as efficient big data learning and capturing complex patterns.

Machine Learning is a broad field that includes many types of knowledge learning, which correlate to many famous AI techniques such as deep learning and generative AI. The scope of this work uses generative AI models, a subset of machine learning models, to improve the AMIE's rule mining algorithm. The following section explains generative AI and how it differs from knowledge-based AI, which uses rules to make predictions.

2.4 Generative AI and Large Language Models

Generative AI is a sub-field of AI and machine learning that has received much attention in the past few years with the release of OpenAI's Generative AI model GPT-3. It uses computational techniques to generate new content from training data rather than traditional knowledge-predicting mechanisms in knowledge-based systems that rely on rules and knowledge bases to predict new knowledge. Generated content includes text, images, audio, and video. In the context of this work, Generative AI models are used to process and generate text, which uses natural language processing as the core component to capture the context of text and generate new text. This section describes Generative AI and highlights its famous fields, such as Large Language Models (LLMs) and natural language processing.

2.4.1 Natural Language Processing

Natural languages are essential prerequisites for writing text and transferring information knowledge. With the large volume of natural language text in the world having a significant content of knowledge, and for the sake of improving artificial intelligence, machines must understand natural language text. Natural Language Processing (NLP) is a set of computational techniques that enables machines to represent and analyze human natural languages. NLP aims to automate language-based tasks such as text processing and translation, question answering, and information retrieval. It is not easy for machines to simulate human understanding of natural language text, as humans generally have background information to understand the context of the text, while machines do not have this background. For example, for the text "Sea water is salty", humans can understand the text without effort because their brains contain much information about seas, but machines generally do not have the knowledge to understand this as humans do. Additionally, NLP has other challenges that make developing NLP difficult, such as:

- **words ambiguity:** Many words can have different meanings, depending on the context of the text, i.e., orange, is it a fruit, or color?
- **complex syntax and semantics:** complex grammatical structures of languages are difficult to interpret
- **contextual understanding:** machines don't have the background and implicit knowledge of the text

NLP Approaches

Natural Language Processing (NLP) has evolved through various approaches, such as rule-based, statistical, and deep learning-based methods. Each approach has advantages and limitations, which determine the suitability of each approach for different NLP tasks.

Rule-Based Approaches (Symbolic AI). These approaches aim at manually defining the syntax and semantics of each language through linguistic rules and dictionaries, which are then used to analyze text. Popular methods include parsing grammar using predefined grammar rules, regular expressions, and pattern matching, and, more importantly, constructing knowledge graphs to represent relations between words (concepts). The last approach is considered a lexical database such as WordNet and FrameNet.

Machine Learning Approaches. Instead of using rules, probability and data-driven learning using statistical algorithms can be used to find relationships between words. They are used in n-Gram Language models to predict the next word based on the previous n words of the text or in Hidden Markov Models (HMM) probabilities to model word sequences. Other approaches include latent semantic analysis (LSA) and probabilistic context-free grammar (PCFG).

Neural NLP. This is the recently dominating approach of NLP. It leverages neural networks to enable machines to automatically simulate human understanding and generate text by learning hierarchical language representations. Popular methods include Recurrent Neural Networks (RNN) and transformer models, which use self-attention mechanisms to process entire sentences simultaneously, such as BERT and GPT models. Section 2.4.3 describes the transformer models in detail, as they are the core of Large Language Models used in this work.

2.4.2 Extractive AI

In information systems, the concept of information retrieval (IR) refers to the process of searching for relevant information in large datasets and retrieving ranked documents (texts) in which this information appears. It focuses on finding valuable information based on the user's query, i.e., search request. IR generally serves queries in natural language by processing these queries into a format understandable by the system using text preprocessing techniques such as stemming, spelling corrections, and stop-word removal to create tokens, which are the smallest units of meaningful text. Traditional IR

creates indexes from texts by applying text preprocessing techniques and associating the resulting tokens with lists of documents in which they appear. A user query is translated into tokens to retrieve all documents in which each token appears. After that, a ranking algorithm is applied to rank the relevancy of each document based on the user query. Popular ranking algorithms include TF-IDF, BM25, Vector Space Model, and AI ranking such as BERT and GPT.

Extractive AI is a broad AI technique concerned with retrieving, extracting, and summarizing information from structured and unstructured data. It is a general concept that includes various sub-fields such as Information Extraction (IE), Extractive Summarization, and Extractive Question Answering (QA). Information Extraction (IE) is the process of retrieving relevant information from documents. It is not the same as information Retrieval (IR), which retrieves relevant documents [GW98]. Popular applications of IE are Named Entity Recognition (NER), event extraction, and financial document analysis, while a famous IR example is search engines such as Google Search. Extractive Question Answering, however, is another task of extractive AI that extracts the most relevant information from a document to answer a user query given in natural language, i.e., a question. Famous applications for Extractive QA are Chatbots and AI-Powered search engines.

Training Extractive AI models depends on the task, i.e., IR, IE, or extractive QA. The training of extractive AI models is, in general, similar to training classic machine learning models using different learning techniques such as Supervised Learning, Unsupervised Learning, Fine-Tuning Large Language Models (LLMs), and applying model evaluation using measures like precision, recall, MRR, and F1 Score.

Generative AI vs. Extractive AI

Unlike Generative AI, Extractive AI does not generate new content; instead, it finds and extracts relevant text, phrases, and entities from the data. Extractive AI applications include traditional search engines, text summarization applications such as selecting sentences from text, and image processing applications such as facial recognition.

Generative AI can generate original content such as text, images, code, music, etc. For example, the user question *Who founded SpaceX?* is answered differently by Generative AI and Extractive AI applications:

Type	**AI Response**
Generative AI	"Elon Musk founded SpaceX in 2002 with the goal of reducing space travel costs."
Extractive AI	"SpaceX was founded in 2002 by Elon Musk."

Table 2.2: Comparison of Generative AI vs. Extractive AI Responses

As shown in Table 2.2, Generative AI paraphrases or generates a sentence that includes the information, while Extractive AI directly extracts the information from a document.

This work facilitates the Generative AI power to improve the rule-mining process of AMIE in many aspects, such as generating logical rules (new content) for given prompts.

2.4.3 Transformers

Generative AI generates new content based on user prompts: text, image, audio, and video. To generate new content, Generative AI requires a deep understanding of the user's prompt context to generate relevant content. This is where Natural Language Processing (NLP) comes in as the core component that Generative AI uses to understand the user's prompt and generate textual content. One of NLP's most popular approaches to capture the context from the text is the transformers model, which recently received much attention. This section describes Transformer models, the building blocks of Large Language Models.

Traditional sequence models such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory Networks (LSTMs) used in NLP process data sequentially, where each token is the text is processed one at a time in order, making it slow and difficult to parallelize. Therefore, the concept of The Transformer has emerged [Vas17]. The Transformer is a model architecture that relies entirely on self-attention mechanisms to capture global dependencies between inputs and outputs, i.e., between the textual input text and the output. The output of the Transformer model depends on the task. In contextual understanding tasks, the output is embeddings or a probability distribution over the possible meaning of the text [DCLT19].

Model Architecture

The Transformer defines two main parts: an encoder to process input data and a decoder to generate output data. Encoder and decoder have the same architecture consisting of multiple neural layers *(N = 6)*, each containing two sub-layers: (1) a Self-Attention Mechanism to understand the relationships between words and capture contextual information and (2) a Feedforward Neural Networks to apply transformations. Additionally, after each sub-layer, there is an Add layer and a Normalize layer, called Add & Normalize layers. They aim to connect residuals and normalize the outputs of each layer and sublayer in the transformer, thereby preventing information loss. Figure 2.1 the architecture of Transformers [Vas17].

Self-Attention Mechanism. To capture the context of the text, this mechanism allows focusing on different sequences of words without considering their positions in the text. This works by first tokenizing the text into tokens and computing three vectors for each token:

- **Query Q:** represents the question the token asks
- **Key K:** represents how relevant the token is for a query
- **Value V:** contains the information passed to the next layer

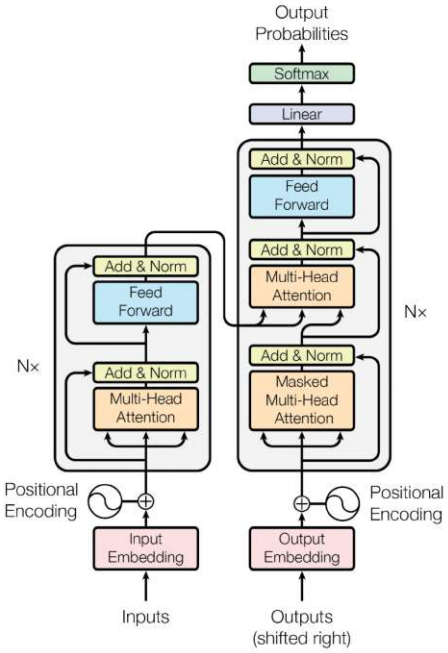

Figure 2.1: The Transformer Model Architecture

These vectors are used to compute attention scores, determining how much attention each word should pay to other words in the sentence. This means that the attention of a word in the text is determined by the attention of the other words in the text to this word. The attention score of a word, represented by the three vectors **Q**, **K**, and **V**, is computed on a set of queries, keys, and values simultaneously and packed together into the three matrices Q, K, and V. The attention score is computed by the attention function, which is defined as [Vas17]:

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

where d_k is the dimension of the Key vectors, K^T is the transposed matrix **K**, and the softmax function is to normalize the scores. Words with a higher attention score get more weight in the neural network than words with lower attention scores.

The mentioned self-attention mechanism is called ***Single Dot-Product Attention***, the core Transformer's attention mechanism. However, it only captures one aspect of relationships, i.e., focuses only on one pair of words that pays the highest attention to each other. Therefore, ***Multi-Head Attention*** has been introduced, which performs multiple attention operations in parallel [Vas17]. It can capture multiple aspects of relationships, such as subject-verb connection, object-adjective relation, and pronoun

resolution. For example, for the sentence *"The cat sat on the mat because it was soft."*, *Multi-Head Attention* would connect the subject *cat* with the verb *sat* and link the object *mat* with the adjustive *soft*, while *Single Dot-Product Attention* would only focus on the relationship between the two words *cat* and *sat*.

Feedforward Neural Network (FFN). In addition to the self-attention sub-layers, each layer in the encoder and decoder contains a fully connected feed-forward network, which is applied to each token of the input text independently and identically at every position in the sequence. This network uses self-attention as input to apply non-linear transformations to the word representations at each layer to produce a new representation of the word, which is then the input to the next Transformer layer, where self-attention and FFN are applied again. This enhances the word representations beyond self-attention by refining the model's understanding [Vas17].

Transformer Applications

Transformers have shown remarkable results in NLP by providing context-aware language understanding. Beyond NLP tasks, Transformers have also performed well in other data processing tasks.

Natural Language Processing. The Transformer's ability to capture the context from text allowed them to be used in most NLP tasks such as machine translation, text summarization, sentiment analysis and text classification, and question answering and information retrieval. The most famous application of Transformers is in Large Language Models used in text generation tasks and chatbots such as ChatGPT applications.

Multimodal Transformers. Convolutional neural networks (CNNs) have been leading the area of computer vision by capturing the features of images efficiently, which includes image recognition and object detection, image captioning, and image generation. Multimodal Transformers extend the standard Transformer's model architecture to "learn representations directly from unaligned multimodal streams" [TBL+19]. Here, the challenge is the unaligned data that comes from different sampling rates of different modalities (language, video, audio). Therefore, they use a cross-modal attention mechanism that allows each modality to adapt its representation based on others without explicit alignment [TBL+19]. Multimodal Transformers have surpassed CNNs' ability to capture the context from different modalities. Popular applications of multimodal Transformer are ChatGPT-4o and Gemini (see Section 2.4.4).

2.4.4 Large Language Models

A Large Language Model (LLM) is a Transformer-based model that processes and generates human-like content using large data. They are trained on billions of words to process the user's prompt and capture its context to perform a task. LLMs do not need to be fine-tuned to a specific task, as they can generate new relevant content directly based on the context of the user's prompt. Fine-tuning can still be performed to improve a

model's performance in a specific task, such as generating code, translating text, creating images and other specific tasks.

LLMs are pre-trained models; they learn general language patterns using unsupervised learning techniques such as the Masked Language Model (MLM) and Autoregressive Prediction. Additionally, reinforcement learning can improve the quality of LLM-generated content and reduce biases by facilitating human feedback (RLHF).

Popular Large Language Models

BERT. Bidirectional Encoder Representations from Transformers (BERT) is a Transformer-based model that uses the encoder part of Transformer model architecture to capture the context of the text by looking both ways of a sentence, i.e., left-to-right and right-to-left. For example, the work *bank* can refer to the financial institution, the side of a river, or the sitting object. In the sentence *"She sat by the bank of the river"*, BERT would look at the left context *"She sat by the"* and at the right context *"of the river"* to understand that the word *bank* refers to a riverbank in this sentence, not to a financial institution. This means that BERT reads both the left and side context of each word before assigning the correct meaning to a word [DCLT19].

Bert uses a pre-trained Transformer model that was created using special unsupervised learning techniques:

- **Masked Language Model (MLM):** random words in the text are hidden, and BERT must predict them (e.g., "*the cat sat on the [MASK]*", BERT predicts *mat*)
- **Next Sentence Prediction (NSP):** BERT predicts whether one sentence follows another to capture sentence relationships in the text.

The pre-trained model is often fine-tuned later for specific NLP tasks, such as Extractive AI tasks such as Named Entity Recognition (NER), Question Answering, Text Classification, etc. [DCLT19].

GPT. Generative Pre-trained Transformer (GPT) is a Generative AI Transformer-based model that uses deep learning techniques to generate new content based on an initial prompt. They use the encoder part of the Transformer model and are trained using the Autoregressive Prediction technique, an approach to learning speech representation by predicting a future frame given the past frames [Rad18] [YYC+22]. GPT models were introduced by OpenAI and are used within the ChatGPT application in question-answering tasks, with the famous release ChatGPT-3 and ChatGPT-4. GPT models have been continuously updated, with releases such as GPT-4o, which is built based on a multimodal Transformer to allow processing of other input data types.

It is important to mention that GPT is a statistical model. It generates sequences of words (and other data) starting from the prompt by making probability-based predictions using the self-attention mechanism and deep learning techniques in the Transformer

architecture. In other words, they predict the most likely answer to a prompt based on the training data. Hence, the quality of the responses depends on the quality of the training data and the training process in general. GPT models excel in pattern recognition and contextual understanding, and they contain pre-trained knowledge embedded as billions and trillions of parameters (model weights) in the model.

Gemini. Google DeepMind's Large Language Model has emerged as a versatile competitor to OpenAI's ChatGPT but with broader capabilities. It is built on a Transformer-based architecture with multimodal enhancements, which allows it to process a wide range of inputs such as text, audio, image, video, and code. One key feature of Gemini is the Retrieval-Augmented Generation (RAG), which uses information retrieval to generate factually grounded output text. Unlike GPT models that use the decoder part of the Transformer architecture, Gemini uses a multimodal Transformer, an advanced AI model that integrates multiple modalities (data types) in a unified framework, allowing for better reasoning and understanding across different data types. These Transformers use cross-attention to capture the context of different types of information, e.g., could associate spoken words in an audio input with images. For example, Gemini could analyze an image and generate a description for it.

CHAPTER 3

Related Work

This chapter presents previous work done in the field of rule mining, as well as a related approach to this thesis.

3.1 LLM-based Rule Mining

LLMs have shown outstanding performance in natural language processing and can be used in various applications. *ChatRule* is an LLM-based rule generator for mining logical rules over knowledge graphs, which uses the semantic information of a knowledge graph to create prompts for the LLM to generate rules. *ChatRule* ranks the generated rules by their rule quality, which is estimated based on the facts on the knowledge graph. Traditional rule miners such as AMIE [GTHS13] and ontological pathfinding [CGWJ16] aim at discovering co-occurrences of frequent patterns in the KG to build logical rules but are still limited due to exhaustive enumeration of relations. Section 4 describes AMIE in more detail. In contrast to traditional rule miners, *ChatRule* leverages the LLM to generate rules rather than performing an exhaustive enumeration of relations.

3.1.1 Generating Rules

ChatRule generates horn rules, which are rules containing logical implications in the form of Horn clauses, as explained in Section 2.3). For example, the following rule

$$GrandMother(X, Y) \implies Father(Z, Y) \wedge Mother(X, Z)$$

denotes that the mother of person X's father Y is the grandmother. This section describes how *ChatRule* generates this type of rule.

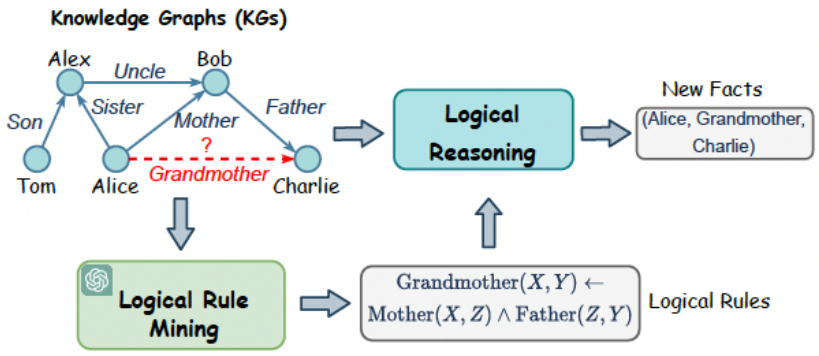

Figure 3.1: ChatRule Concept [LJX+23]

Rule Sampler

A breadth-first search sampler is implemented to create instances of logical rules by sampling *closed-paths* in the KG. In other words, each KG relation, represented as triple $< Subject, Predicate, Object >$, is used to create instances of logical rules by searching for *closed-paths* in the KG. The constants in these instances are then substituted with variables to build candidate rules, which are then fed to the LLM to facilitate the rule generation, referred to as rule samples.

LLM-based Rule Generator

The rule generation using the LLM focuses on the head atom of each rule created by the rule sampler. A prompt is crafted to provide information about the context and a clear task that includes the head atom of each rule sample. Due to the large number of rule samples, sending them all at once to the LLM is infeasible. Therefore, they are divided randomly into subsets, where each subset is included in the prompt and sent separately to the LLM.

3.1.2 Ranking Rules

The rules generated by the LLM are ranked using a scoring mechanism that integrates rule quality metrics, such as confidence and support, and LLM-based evaluations. This mechanism identifies high-quality, logically valid, and semantically meaningful rules. The ranking process involves multiple evaluation criteria. First, ChatRule computes confidence based on knowledge graph facts and introduces semantic plausibility scoring by prompting the LLM to assess the rule's logical soundness and commonsense validity.

The ranking mechanism discards rules with low confidence scores and prioritizes those with higher coverage and applicability. The framework aims to balance precision and recall, ensuring that extracted rules are generalizable and reliable.

3.1.3 Results

The results highlight that ChatRule produces semantically rich and generalizable rules. ChatRule utilizes LLMs to infer deeper commonsense relationships and generate rules that align with human reasoning. This allows ChatRule to mine rules that are not only structurally valid but also semantically meaningful. Additionally, ChatRule handles missing information by utilizing LLM-based reasoning, making it more robust in handling open-world scenarios than traditional rule mining techniques. This leads to higher rule coverage and enables ChatRule to generalize beyond the data in the knowledge graph.

When compared to AMIE+ [GTHS13], ChatRule achieves higher performance in terms of rule interpretability and coverage, as the generated rules can be easily read by humans. Additionally, ChatRule extracts broader and more flexible logical patterns, while AMIE+ is constrained by its reliance on structural dependencies within the knowledge base.

Despite its advantages, ChatRule has certain trade-offs regarding computational overhead due to the reliance on LLM inference. This makes ChatRule slower than AMIE+ for large-scale rule mining. additionally, ChatRule's dependency on multiple LLM calls limits its performance on large knowledge graphs. This makes ChatRule more suitable for quality-focused rule mining rather than high-speed, large-scale processing.

3.2 Traditional Rule Mining

In the context of this thesis, traditional rule mining refers to applying various non-LLM-based techniques to extract rules from knowledge bases, which includes AMIE, as detailed in Chapter 4. This section describes different traditional rule mining approaches.

3.2.1 Inductive Logic Programming (ILP)

Inductive Logic Programming (ILP) is a subfield of machine learning that combines principles from logic programming and inductive inference to construct logical rules from the data in a knowledge base. Specifically, the positive and negative examples of a KB are used to learn rules using first-order logic (FOL) to achieve structured reasoning on data [GTHS13] [MR94]. First-order logic is an expressive formal system used to represent and reason about knowledge [Bar77]. Horn rules are a special type of first-order rules, as described in Section 2.3,

3.2.2 Sherlock

Sherlock [SDEW10] is an unsupervised ILP-based system that learns first-order horn clauses from extracted facts using probabilistic graphical models (PGMs), i.e., probability theory and graph theory. Sherlock is designed to address inconsistencies in knowledge bases by incorporating a preprocessing step that filters noisy facts and avoids longer rules to enhance inference accuracy. Additionally, it uses statistical significance and relevance

heuristics to guide the rule mining process, thereby ensuring that the learned rules are both meaningful and practically useful.

As Sherlock is an unsupervised rule learning technique, it does not require labeled data, making it adaptable to various domains. However, the extensive filtering and inference processes can make Sherlock slower than other rule mining systems, especially on very large KBs. While it excels in noise handling and precise rule generation, its performance can lag when faced with the vast scale of modern knowledge bases, where more efficient, scalable methods like AMIE or RudiK might be preferred [SDEW10].

3.2.3 RudiK

RudiK is a traditional rule miner designed to mine both positive and negative rules in knowledge bases. It generates explicit counterexamples by applying the Partial Completeness Assumption (PCA) and other sampling techniques. For example, when generating counter-facts for a relation like hasChild, RudiK samples individuals who are not children of a given subject but are children of someone else, thereby creating semantically relevant negative examples. This approach enables RudiK to generate a more balanced set of training data, which enhances the quality of the mined rules. RudiK uses a greedy heuristic that iteratively selects the most promising rules in terms of coverage rather than exhaustively searching all rules that do not meet the confidence threshold criteria. This makes RudiK efficient and suitable for prediction tasks. Despite its benefits, it does not guarantee the discovery of all valid rules in the data. Its focus on balancing positive and negative evidence allows it to achieve good performance even on complex KBs, but it can sometimes lead to incomplete results compared to exhaustive methods like AMIE3 [OMP18].

3.2.4 Evoda

Evoda is a rule mining algorithm based on genetic logic programming. It extends the rule hypothesis space from path rules to the more expressive Datalog rule space. It uses genetic programming techniques such as mutation, crossover, and selection to efficiently explore a vast rule space with minimal syntactic constraints, i.e., rule length. Just like AMIE, the algorithm operates under the Open World Assumption (OWA), which acknowledges the inherent incompleteness of KGs and offers a more suitable framework for reasoning in such environments.

Evoda performance evaluation shows that it can mine high-quality rules quickly compared to traditional rule mining methods. Through a series of experiments on real-world KGs, they show that Evoda outperforms state-of-the-art models, both rule-based and embedding-based, in terms of precision and prediction accuracy. The results confirm that Evoda improves the scalability, efficiency, and effectiveness of mining rules for KG completion tasks, making it a competitive alternative to existing traditional rule mining approaches [WSS+22].

CHAPTER 4

Traditional Rule Mining Using AMIE

Rule mining is a data mining technique that involves finding patterns, hidden knowledge, and relationships from data. Rule miners extract logical rules from knowledge bases (KBs) to describe a knowledge base. Rule miners typically use a mining algorithm to extract rules from large knowledge bases and evaluate these rules based on specific metrics against the knowledge base being mined.

While most mining algorithms today deliver near-perfect rules, they cannot often find rules outside the KB because they heavily rely on the facts in the KB as the primary source of knowledge. The time and space complexity of these algorithms increases with the size of the knowledge base, making it harder to mine rules from large knowledge bases.

AMIE is a rule miner inspired by association rule mining that mines horn rules from a given knowledge base. This chapter will explain AMIE's algorithm in detail and address the drawbacks of its latest version, AMIE3.5.

4.1 General Concept of AMIE

AMIE is a rule miner that mines horn rules out of large knowledge bases. Given a KB in turtle format (ttl), an RDF-style triples format, i.e., <Subject> <Predicate> <Object>, AMIE can extract rules that describe the data in the KB. These extracted rules:

- can be applied to the KB data to derive new facts, making the KB more complete.
- can be used to find errors and deviations in the data.

- can be used to enhance reasoning on the KB.
- describes the regularities of the KB data.

4.2 Algorithm

Before the mining algorithm starts, AMIE processes the inputted KB by loading it into memory, and then it starts the mining algorithm, which can be divided into three main phases: initialization phase, evaluation mode, and refinement phase. The mining algorithm works by dequeuing a candidate rule from an initialized queue created in the initialization phase, evaluates it based on the quality metrics in the evaluation phase to decide whether this candidate rule can be outputted or not, and eventually refines it in case the rule is not fit for the output, i.e., does not meet the quality metrics thresholds. Otherwise, if the candidate rule passes the thresholds, AMIE adds it to the output. In the refinement phase, the previously dequeued candidate rule discarded by the evaluation phase is used to create new candidate rules that are then queued to be processed in later iterations. The algorithm keeps track of the refinement rules chain. This section will first describe how the KB is loaded into the memory, then explain in detail the 3 phases of the mining process.

4.2.1 In-Memory Knowledge Base

This is the first phase when running AMIE. Here, the knowledge base in turtle (ttl) format inputted as a program argument is read, analyzed, and then stored in memory. This results in an in-memory database that can be queried efficiently during mining.

The KB inputted contains facts as triples in the form <entityX> <relationN> <entityY>, corresponding to the RDF subject, relation, and object, respectively. Each fact is an atom that can be referenced in datalog format as '$relationN(entityX, entityY)$'. AMIE creates indexes for each fact, meaning that every fact is indexed by subject (S), object (O), relation (R), and by relation-subject and relation-object pairs. This leads to 6 indexes: SRO, SOR, RSO, ROS, OSR, and ORS. These indexes store the entities and relations as the primitive type *int*, which allows for reducing the memory usage to the extent possible and loading large KBs into the memory. This makes duplication and equality checks, hash computations, and other operations on the in-memory database much faster and more compliant with the RDF engine serialization formats.

After the KB has been successfully loaded into memory, AMIE initializes a miner instance from the program arguments and configures the mining algorithm using the proposed parameters. Additionally, a mining assistant instance is instantiated to perform different operations within the mining process, such as quality metrics checks and refinement operations, which will be explained in the subsequent sections.

4.2.2 Phase 1: Initialization phase

At this point, the in-memory database is ready for querying, and the mining assistant is instantiated. Therefore, the mining process can start with the first phase, which is responsible for initializing a queue consisting of candidate rules to be evaluated and refined later.

In this phase, AMIE iterates over all relations in the KB and builds candidate rules that consist of simple implications of each relation. This means that if the KB contains N relations, then the initial queue will contain N candidate rules in the form:

$$true \Longleftarrow r(X, Y)$$

These simple implication rules are used to initialize the queue, and most will likely be discarded within the main mining loop. Still, they are essential for building new candidate rules in the refinement phase.

4.2.3 Phase 2: Evaluation phase:

One characteristic of AMIE is the quality metrics proposed and used to evaluate the rules in the mining process. If a rule passes the evaluation, referred to as AMIE evaluation in this work, it will be added to the result set of the mining process. Otherwise, the rule is discarded, i.e., not added to the result set, but will be refined if it passes two additional checks. The refinement phase is explained in the next section.

Pruning Strategies

The search space is the total set of possible logical rules that can be built from the atoms of the KB to be mined. The number of possible rules grows exponentially with the number of atoms in the KB, which would be infeasible to process using reasonable resources. Theoretically, most rules in the search space will be discarded in the evaluation phase because not all atoms can be combined to form a rule with valid logical connections and contextual relevancy. Therefore, most rule miners define a pruning strategy to reduce the number of rules to be discovered and evaluated, keeping resource utilization reasonable. In AMIE, the pruning strategy decides whether a rule can still be refined, i.e., use it to build new rules or be discarded or added to the output.

AMIE3 uses multiple pruning strategies based on quality metrics and rule properties, defined in Section 2.3, to efficiently explore the search space. The pruning is based on thresholds, i.e., if the rule property does not meet the threshold constraint, then it is not further refined and will be either discarded or added to the output. The pruning strategies that AMIE3 uses to decide whether a rule can be outputted are as follows:

- rule is **closed**

- **Minimum Head-coverage Threshold:** ensures that rules covering a small fraction of the KB facts (threshold) are not included in the output set to prevent the algorithm from outputting insignificant rules.
- **Minimum PCA-Confidence Threshold:** the PCA-confidence of a rule must meet the threshold to avoid making incorrect predictions by this rule.
- **Parent Rules Confidence Pruning:** the PCA-confidence of a rule must be higher than the PCA confidence of all previously mined rules with the same head atom and a subset of its body atoms

The pruning strategies also ensure that AMIE does not refine bad rules to avoid unnecessary computations done by unuseful refinements. Therefore, AMIE3 defines pruning strategies to reduce the search space size as follows:

The AMIE evaluation of a rule consists of calculating the quality metrics of a rule and verifying that the rule meets certain criteria. The main criteria that a rule must meet to be added to the result set are:

- **Minimum Rule Length Threshold:** rules that meet the rule length threshold are not further refined.
- **Rule confidence can still be improved:** If adding a new atom will not increase the confidence, the rule is not refined.
- **Perfect Rule:** If the PCA-confidence of a rule is 100%, it cannot be improved further, so it will not be refined and will be added directly to the output.

Quality Metrics

The horn rules properties defined in Section 2.3 are crucial for AMIE evaluation to decide whether a rule can be outputted or not and whether they can be further refined, as explained above. For every rule outputted by AMIE, the following quality metrics are shown so the user can manually inspect the mining result.

- **Head Coverage:** the proportion of facts in the KB the rule covers, i.e., can predict.
- **Standard Confidence.** the confidence score under Closed World Assumption (CWA) to measure the accuracy of a rule.
- **PCA Confidence.** The outputted rules are ranked by the PCA-confidence score and are, unlike the standard confidence, a better approximation to the real-world rule quality

- **Positive Examples.** number of *true* predictions made by the rule and exist in the KB.
- **Body size.** number of times the body of the rule is matched in the KB.
- **PCA Body size.** number of expected cases where the head of the rule is missing but is covered by the PCA assumption.

Additionally, a rule's *support* is always calculated in the mining process and used in the head coverage calculation. It is not outputted as a quality metric, but it is better to measure the significance of a rule by the head coverage metric.

4.2.4 Phase 3: Refinement phase:

Rules discarded by AMIE evaluation, i.e., do not pass the pruning strategy to output a rule, are refined. The refinement of AMIE, referred to as AMIE refinement in this work, uses the so-called refinement operators to derive new rules (children) from the current rule (parent) in the iteration. It builds new rules by creating a new rule for each possible atom in the KB that can be added to the body of a rule.

The refinement operators are responsible for deriving these new rules. The three refinement operators used in AMIE are described below.

Refinement Operators

AMIE defines three refinement operators for exploring the search space by iteratively extending rules. This means each refinement operator is applied to a rule one at a time to derive new rules from the current rule. The refinement operators all add a new atom to a rule, but they differ by how each handles the arguments of new atoms, defined as follows:

- Add Dangling Atoms: adds a new atom with one new variable and one shared variable used in another atom of the rule.
- Add instantiated Atom: adds a new atom that uses an entity and a shared variable
- Add closing Atom: adds a new atom with two shared variables used in other atoms of the rule

The new rules derived by the refinement operators all produce connected rules. These rules are then added to the queue to be processed in later iterations, i.e., they will be evaluated and refined. Table 4.1 shows how each refinement operator handles the arguments of new atoms to be added to a rule.

The refinement operators build almost all possible rules that can be mined from the KB. This results in a huge queue to be processed, making AMIE refinement the bottleneck of

Operator	New Atom Argument 1	New Atom Argument 2
Add Dangling Atoms	new variable	shared variable
Add instantiated Atom	entity	shared variable
Add closing Atom	shared variable	shared variable

Table 4.1: Refinement operators arguments handling

the AMIE mining algorithm. Having all possible rules in the queue means that AMIE will eventually find all possible valid rules to be mined.

All refinement operators need to choose a KB relation to add the new atom to the rule being refined. This happens using Projection Queries, as explained in the next section.

Projection Queries

AMIE uses projection queries to retrieve only relations that fulfill the head coverage constraint. This is crucial to avoid enumerating all possible combinations of atom conjunctions, to improve the refinement runtime, and to reduce memory consumption. The queries are executed on the in-memory database, which allows for data retrieval in a constant amount of time. Section 4.3.1 explains the performance details of AMIE.

4.3 Technical Aspects of AMIE

4.3.1 Performance

One major challenge of rule miners is their performance and runtime. Rule miners should be able to find rules on large KBs efficiently. It has been shown in [LGS20] that AMIE runtime is minimal compared to other rule miners. This section will describe how AMIE maintains a good runtime performance.

In-Memory Database

In [GTHS13], it has been shown that using a SQL or SPARQL engine to query the KB to be mined is infeasible, and it will take several minutes to run queries, which is very time inefficient considering the case that queries are to be executed within the mining process. In its early versions, AMIE implemented a ByteString-based in-memory database with extensive indexing to address this issue, which allows for retrieving the instantiations of an atom in constant time. In later versions, this BystrString-based in-memory database is migrated into an Integer-based in-memory database, as explained in Section 4.2.1. This implementation not only boosts the performance using low-latency memory operations but also minimizes the query processing time. It has been shown in [LGS20] that the Integer-based in-memory database implemented in the latest version of AMIE, AMIE3, reduced the memory consumption by 3 GB in most cases compared to the old ByteString-based in-memory database implemented in the former versions of AMIE, namely AMIE, and AMIE+.

Refinement Performance Optimization

The refinement phase is the bottleneck of the mining algorithm. Here, projection queries are executed to retrieve the relevant relations to be used in the new atom for a rule. In addition to boosting the performance using the projection queries, AMIE implements a technique for further reduction of the refinement runtime by defining constraints for controlling the application of refinement operators instead of blindly applying all operators.

In addition to this, the refinement runtime is further reduced by the concept of 'perfect rule', which makes use of the definition of PCA confidence, which states that a rule cannot achieve a PCA confidence higher than 100%. A rule that achieves 100% PCA confidence score will be added to the output and its refinement will be stopped.

Other Optimizations

Another big part of the mining process is the computation of confidence scores because at this part the number of instantiations of the body in a rule is calculated. This is done by performing join operations between the relations, which is computationally expensive considering large KBs. To address this issue, AMIE implements an approximation based on statistics to reduce the number of joins between two relations.

To further improve the performance, AMIE parallelizes the whole process by firing as many threads as possible in parallel, that is, as much as the host system allows.

4.4 Drawbacks of AMIE

Deep experiments have been made on AMIE to show the algorithm's effectiveness on medium-sized and large KBs. The experiments included running AMIE with different configurations and comparing the results of AMIE, RudiK, and Ontological Pathfinding when running with their default configurations on the same dataset. Furthermore, the experiments used five different datasets as input to AMIE and described the impact of the different performance optimizations on the runtime. It has been shown that AMIE in its last version [LGS20] achieves the best runtime over the other rule miners. Additionally, the number of mined rules by AMIE3 outperformed both OP and RuDiK.

Despite the upsides of AMIE3, there are still some drawbacks described in this section.

4.4.1 Contextual Rule Evaluation

AMIE evaluation relies on computing different quality metrics to measure the quality of the mined rules. As mentioned in Section 4.2.3, AMIE decides if a rule can be outputted based on three criteria. Otherwise, a rule is discarded and refined to generate new rules in the AMIE refinement phase. Many discarded rules that are closed have a low PCA confidence, but they still could have a valid context. In other words, if there are not enough *true* predictions in the KB that the rule can predict.

Such rules can still be fit to be added to the output, but they do not pass AMIE evaluation due to their low PCA confidence score.

4.4.2 High Resource Utilization

AMIE loads the whole KB into the memory and creates 6 indexes from the KB to speed up the mining process. This leads to high memory consumption which can be up to 500 GB depending on the KB size. This can be a limiting factor when access to such high-memory resources can not be provided.

There are many aspects of the mining process that can be optimized to reduce the runtime and CPU usage even further. For instance, the initial queue of AMIE that consists of simple implications of head atoms can be optimized by starting with meaningful rules with valid context that are then evaluated and refined by AMIE. Additionally, the refinement operators generate so many rules with invalid context that are then pruned away either by the head-coverage criteria or by AMIE evaluation. Eliminating these rules can reduce the runtime of AMIE by avoiding unnecessary computation.

4.4.3 Domain Knowledge

The rules mined by AMIE strictly rely on the KB facts. The atoms of a rule are formed from KB relations, which is the idea of rule mining. However, AMIE cannot build rules from a mix of KB relations and relations from the same domain that are not found in the KB. This could result in a broader domain coverage by the mined rules that can be then used to add new facts to KB, enriching it even further and less strictly.

CHAPTER 5

Approach

To address the drawbacks of AMIE's algorithm and to improve the quality and quantity of the rules outputted by AMIE, discussing the research questions in Section 1.2, this work proposes the usage of a Large Language Model (LLM) that facilitates its contextual reasoning and natural language processing power to suggest, evaluate, and refine rules, enhancing thereby the overall mined result set. This would ensure that the mined rules have a valid context and that rules can be found from outside the knowledge base, thereby introducing new atoms and entities in the KB.

To use the LLM in AMIE, the base code of AMIE3.5 [DIG25] is extended by integrating an LLM client that enables communication with an LLM within the mining process, addressing the issues of the different mining phases mentioned in Section 4.2. This chapter explains in detail the approach adopted to integrate an LLM in AMIE, the implementation of the LLM client and the different LLM tasks defined, the modifications to the AMIE algorithm, and the settings introduced to run AMIE with the LLM integration.

5.1 Idea

AMIE mines rules by trying almost all possible combinations of atoms to build rules in the refinement phase, add them to the queue, and evaluate each rule. Most of these rules are discarded by AMIE evaluation, whereas some might contain a valid context. The base idea is to integrate LLM calls within the three phases of AMIE to perform different tasks, which are then called **LLM tasks**. This work proposes and implements the usage of LLMs in three tasks, which correspond to the 3 phases of the AMIE algorithm and are integrated within these phases.

5.2 LLM Initial Queue Task

As mentioned in Section 4.2, AMIE initializes an initial queue consisting of simple implications of every relation in the KB, meaning that if the KB contains N relations, the initial queue would contain N rules in the form: $true \Leftarrow r(X, Y)$. These rules are then refined within the refinement phase to build other rules using the refinement operators. On the other hand, the LLM can generate rules with a valid context when a list of KB relations is provided, incorporating each relation's contextual information. This functionality is integrated within the initialization phase of AMIE to create a prompt for the LLM, wait until the LLM responds, and finally, parse the LLM response to build the objects of the generated rules that AMIE requires. Algorithm 5.1 shows the steps in the initial queue task. AMIE provides the list of KB relations after loading the KB into its in-memory integer-based database, as described in Section 4.3.1. All relations are then unmapped to retrieve the string representations containing the required contextual information and append them to a prompt, as explained in the next section.

The adopted approach sends one request by default to the LLM to build as many rules as possible using contextually relevant relations from the list provided. Due to model optimization, fine-tuning, request limitations, and other factors, the LLM might only generate a handful of rules from the given list of relations (see Chapter 7). To ensure the LLM generates as many rules as possible and uses all statistically possible relevant relations, the request is repeated N times, where the LLM generates N lists of rules in N responses, allowing for a statistically better rule generation. The following section explains the request repetition concept, which includes using the rules suggested by the LLM.

Algorithm 5.1: LLM Initial Queue Task Algorithm

Input: A list of strings representing input rules
Output: A list of rule objects

```
1  resultSuggestedSet = [];
2  Build the prompt and configure the request ;
3  for i from 0 to repetitionNumber do
4  |   Throttle LLM requests;
5  |   jsonResponse = generate response by LLM;
6  |   generatedRules = parseLLMGeneratedRules(jsonResponse);
7  |   foreach rule in generatedRules do
8  |   |   add the rules to resultSuggestedSet based on the selection mode:
   |   |     UNIQUE, DUPLICATE, or ALL
9  |   end
10 end
11 return resultSuggestedSet;
```

5.2.1 Prompt

For the initial queue task, the LLM is requested to generate logical rules by combining relations with contextual relevance and introduce correct variability represented by meaningful logical connections through each relation. In the prompt, all KB relations are provided for the LLM to use at the end of the prompt. In the beginning, the prompt contains a short description of the overall task and domain of knowledge and a template for the logical rules. Additionally, clear instructions are given as a list to the LLM so it can combine relations and provide their variability correctly, thereby avoiding generating rules by randomly combining relations without contextual relevancy or combining relevant relations without valid logical connections. The rules generated by the LLM should not have a common pattern, i.e., repeating a relation twice in a rule or showing a pattern in logical connections.

Other instructions are guidelines for formatting rules, as AMIE can only parse strings to rule objects in a pre-defined format. These guidelines state a regular expression to be used for variables and some restrictions on generating rules, such as using only relations from the provided list of relations, which is the main idea of the initial queue task. In addition to guidelines and instructions, a list of example rules is also provided to ensure the rule format and the instructions are exemplified for the LLM. Finally, a clear task is provided at the end of the prompt, which states exactly what the LLM should do.

Section 6.2 shows the detailed prompt for the initial queue task.

5.2.2 Request Repetition

As mentioned earlier, the initial queue task contains the functionality of repeating the request to the LLM N times to allow for a more comprehensive rule suggestion, resulting in N responses. The N responses could contain duplicated rules. An assumption is made that a rule suggested more than once by the LLM could have a relatively high probability of being outputted than the other rules suggested only once. The adopted approach defines different rule selection strategies from the N responses to address and test this assumption as follows:

- **Unique Selection:** Select only the rules not duplicated in N responses.
- **Duplicate Selection:** Select only the rules duplicated in N responses.
- **ALL Selection:** Select all rules in N responses.

Handling the N responses generated by the LLM depends on the chosen rule selection strategy. The duplicate selection strategy is expected to select high-quality rules that are likely to be outputted. In contrast, the unique selection strategy is expected to improve the number of rules to be outputted. Section 7 describes the experiments to test these assumptions.

5.2.3 Integration within AMIE

After the selection of rules is finished, the rules are parsed into rule objects, as described in Section 6.2. These rule objects can then be used in the mining process, i.e., integrated into the initial queue of AMIE, which is initialized using a list of rules called *seed rules*. AMIE's *seed rules* consists of simple implications, as explained in Section 4.2. Two modes *ONLY_LLM*, and *LLM_AND_AMIE* are defined to use these rules within the initial queue of AMIE, which is two of the three modes this work defines to control the integration LLM within the initial queue task (see Section 6.6.2):

- Use only the simple implication rules (mode: ***ONLY_AMIE***)
- Use only the rules suggested by the LLM in the initial queue (mode: ***ONLY_LLM***)
- Use the simple implication rules and add the rules suggested by the LLM to the initial queue (mode: ***LLM_AND_AMIE***)

Choosing between these two approaches is realized using environment variables, i.e., settings read by AMIE at startup. Section 6.6.2 describes the different settings that can be used to control the integration of all LLM tasks within AMIE. Algorithm 5.2 describes integrating the LLM initial queue task within AMIE.

Algorithm 5.2: Integration of LLM Initial Queue Task in AMIE

```
   Input: seedRules, initializationMode
   Output: seedRules
 1 if initializationMode ≠ ONLY_LLM then
 2 |   AMIE Seed Rules Initialization
 3 end
 4 if initializationMode ≠ ONLY_AMIE then
 5 |   suggestedRules ← LLM suggested rules;
 6 |   if initialQueueTask.getMode() = ONLY_LLM then
 7 |   |   seedRules ← suggestedRules
 8 |   end
 9 |   else if initialQueueTask.getMode() = LLM_AND_AMIE then
10 |   |   seedRules.addAll(suggestedRules)
11 |   end
12 end
13 return seedRules;
```

5.3 LLM Evaluation Task

The mining process iterates over the rules in the initial queue, performing AMIE evaluation and refinement for each rule, as explained in Section 4.2. Many rules might have a valid

context but are discarded by AMIE because they do not pass the evaluation thresholds. To incorporate the rule context, LLM can be used to evaluate the context of each rule to *true* or *false*. In other words, the LLM can decide if the contextual information of each atom in the rule is relevant to the other atoms and if the logical connection between the atoms, realized through shared variables, is valid.

To integrate this approach within AMIE, the AMIE's evaluation result is considered a primary criterion for outputting a rule, and the LLM evaluation result for the same rule is an additional and more decisive criterion for accepting rules discarded by AMIE. Therefore, if AMIE decides that a rule is not fit for output, it will be contextually evaluated by the LLM. If the LLM evaluates the rule context to *true*, it will be added to the result set, overriding the AMIE evaluation result.

AMIE evaluation is done within the mining process and is considered fast in AMIE's latest version, AMIE3.5. The same does not apply to the LLM evaluation, as the LLM takes a few seconds to evaluate the context of each rule individually, leading to a significant increase in the runtime of the mining process. Considering that the queue of AMIE is huge at the start, and the refinement phase continuously adds new rules to it, making it even larger, evaluating the context of each rule individually by the LLM will become very costly. Considering that most rules in the queue have an incorrect context because of the refinement operators, it is expected that the LLM will evaluate most rules to *false*. Therefore, only rules with more than three atoms are contextually evaluated by the LLM, considering there is no way to tell if any rule with less than three atoms would have a correct context. For instance, a rule with two atoms such as `?a motherOf ?b` ⇒ `?a worksWith ?b` does not contain enough information to evaluate its context. Only rules with two atoms that contain an inverse context, such as `?a motherOf ?b` ⇒ `?b daughterOf ?a` can be evaluated for contextual correctness, but these are only a few cases that AMIE evaluation can handle alone just fine. Rules with more than three atoms infer logical connections between shared variables, providing enough information to evaluate the context of a rule. This approach ensures only meaningful LLM evaluations are made, which reduces the number of requests made to the LLM and, consequently, the costs of using it.

Another issue arises if each rule's context is evaluated separately by the LLM, as this approach can still cause a huge runtime overhead, given that the queue is huge. Additionally, the more relations the KB contains, the bigger the queue is. As the rules whose context is evaluated by the LLM to *true* are added directly to the result set, overriding the AMIE evaluation result, then the LLM integration approach can be optimized by not evaluating each rule separately within the main loop of the mining process, but evaluating all rules discarded by AMIE at the end of the main loop at once. This significantly reduces the request count to the LLM, leading to less runtime overhead. Surely, the LLM will take longer to evaluate N discarded rules than one rule, but this is still a reasonable time compared to sending N requests for the LLM to evaluate each rule separately. Section 6.3 explains the detailed implementation of the AMIE evaluation task.

As mentioned earlier, context evaluation is considered an additional criterion for outputting a rule, which is stronger than the other criteria of AMIE evaluation. This, however, relies solely on the LLM to provide a correct evaluation. Unfortunately, the results of this work in Chapter 7 show that LLM is not always reliable in evaluating the context of rules. Therefore, AMIE quality metrics and additional statistics computed to track the LLM tasks (explained in Section 6.5) are outputted for each rule, so a user can manually inspect the mining result and remove those rules that are unsuitable for the output.

5.3.1 Batching and Throttling Requests

As mentioned above, the approach of the LLM evaluation task is to evaluate the context of all rules discarded by AMIE. The number of these rules can be huge, which collides with the restrictions imposed on using the LLM, particularly the maximum output tokens limit defined by Gemini. Suppose all discarded rules are sent in one request to the LLM for contextual evaluation. In that case, the LLM may generate a partial response, i.e., it may evaluate the first N rules and ignore the rest to comply with the maximum tokens output limit, leading to an inaccurate evaluation. To avoid this behavior, the approach is to send the rules to be evaluated in batches, i.e., sending N rules in one request, then sending the following N rules in the following request, and so on, until the LLM contextually evaluates all rules. This implies sending multiple requests within the LLM evaluation task, which leads to another issue: the LLM requests per minute limitation, which would ignore all requests sent within a minute after M requests have been processed. Therefore, the approach is to throttle these requests, i.e., wait until the minute ends. Algorithm 10.1 shows how to throttle the requests sent to the LLM.

5.3.2 Prompt

For the evaluation task, the prompt must communicate clearly that the LLM should evaluate the context of a given list of rules to *true* or *false*. Additionally, the LLM must know how to evaluate each rule, given that the AMIE rule format is unknown to the LLM. Therefore, the prompt contains a short introduction that shows the template of a rule, along with an example rule and its interpretation, i.e., contextual information about it. This way, the LLM would know how to check the logical connections between the atoms in each given rule and understand the meaning of each atom.

5.3.3 Integration within AMIE

The LLM response on each batch is parsed by matching the rule strings with the rule objects of the batch for those rules whose context is evaluated to *true* by the LLM. Algorithm 5.4 describes the approach for parsing the LLM response, and Section 6.3 describes the detailed implementation of this algorithm. Rules evaluated to *true* by the LLM are added directly to the mining result, and the ones evaluated to *false* are

ignored. Algorithm 5.3 describes integrating the LLM evaluation task within AMIE, which consumes the result of the algorithm 5.4.

Algorithm 5.3: Integration of LLM Evaluation Task in AMIE

```
   Input: batches: batched lists of discarded rules
 1 evaluationObjects = [];
 2 Function EvaluateBatch(batch):
 3 |   Build the prompt and configure the request ;
 4 |   jsonResponse =  generate response by LLM;
 5 |   evaluations = parseEvaluationResponse(jsonResponse,
   |    batch);
 6 |   return evaluations;
 7 foreach batch in batches do
 8 |   evaluations = EvaluateBatch(batch);
 9 |   foreach evaluation in evaluations do
10 |   |   if evaluation.context == true then
11 |   |   |   if evaluation.rule ∉ result then
12 |   |   |   |   Add evaluation.rule to the mining result
13 |   |   |   end
14 |   |   end
15 |   end
16 end
```

5.4 LLM Refinement Task

AMIE refinement is the bottleneck of the mining process. Here, refinement operators add atoms to the current rule being refined to build new rules that are subsequently queued for processing in future iterations. Section 4.2 describes this phase of AMIE in detail. Adding new atoms to a rule that does not get pruned creates many rules, most with incorrect context. This leads to a huge queue of rules to be processed, affecting the mining process's runtime and resource utilization. The LLM can be used in this phase to refine valid rules that pass AMIE evaluation, i.e., outputted rules, based on the contextual information each rule contains and the logical connections between the variables of each rule. In other words, the output rules are sent to LLM to generate new rules, i.e., the LLM refines them. Like the AMIE refinement, LLM refinement discovers new rules, possibly with new atoms from outside the KB, by incorporating domain knowledge, thereby enriching the KB by adding new relations and facts. This cannot be achieved with AMIE refinement, which relies only on the atoms in the KB to build rules.

While the previously mentioned factors contribute to AMIE's performance, it is important to recognize that AMIE's refinement process extracts nearly all possible rules from the knowledge base. Additionally, the LLM refinement leverages the rules added to the

Algorithm 5.4: parseEvaluationResponse(): Parsing Evaluation Task LLM

```
Input: jsonResponse, batchRules
Output: List of EvaluationObjects
evaluations ← ∅;
rulesArray ←array containing rules in string format from jsonResponse;
foreach evaluation in rulesArray do
    contextEval ← evaluation.get("context");
    rule ← AMIEParser.parse(evaluation.get("rule");
    foreach originalRule in batchRules do
        if originalRule.getRuleString() = rule.getRuleString() then
            if contextEval then
                Add evaluation to evaluations;
            end
            break;
        end
    end
end
return evaluations;
```

output to generate new rules in the same domain. Since AMIE refinement occurs after AMIE evaluation in the main loop and the refined rules undergo further assessment in subsequent iterations before being added to the final result set, disabling AMIE refinement is not an option. It remains a crucial step in the mining process for generating new rules and providing the output required for the LLM refinement. Therefore, this LLM task in this work cannot replace AMIE refinement. Instead, it is complimentary to it. The runtime remains reasonable, as the LLM refinement procedure is executed at the end of the mining process and not in the main loop.

5.4.1 Prompt

The prompt template for the refinement task is similar to the template created for the initial queue task, as both tasks generate new rules using the context from the data. Unlike the initial queue task, which strictly uses KB relations to generate new rules, the refinement task uses a list of rules to create new rules with new potential relations by incorporating domain knowledge that matches the domains of the provided list of rules. Therefore, the prompt in this task includes asking the LLM explicitly to incorporate domain knowledge by considering the context of each rule. Additionally, the prompt includes the list of rules outputted by AMIE, referred to as *main output set*. Finally, the prompt retains the same characteristics as the initial queue, specifically the same rule format template, identical instructions and guidelines, and consistent rule examples.

5.4.2 Integrating within AMIE

The LLM refinement task is executed at the end of the main loop and after the LLM evaluation. In other words, the LLM refinement is the last step of the mining process. This allows for a complete mining result before performing LLM refinement. Similar to the LLM evaluation task, the rules outputted by AMIE are batched, and the requests are throttled to ensure the LLM does not cut down responses and that requests are not ignored. After the LLM discovers new rules, they are parsed similarly to those generated by the initial queue task, as described in Section 6.2. The resulting rule objects, however, cannot be added to the result set, as there is no proof of their correctness and quality metrics. Therefore, they are first evaluated using the AMIE evaluation procedure and LLM evaluation. The idea is to rerun the mining process on the newly discovered rules by the LLM, i.e., the initial queue in the second run of the mining process contains only rules created by the LLM refinement task. This includes rerunning AMIE evaluation and AMIE refinement, as they both occur in the main loop. AMIE refinement would still try to add all possible atoms to each rule. Still, the maximum rule length threshold in AMIE configuration will prohibit this because the rules in the queue, i.e., those generated by the LLM, have already reached the maximum rule length threshold, which, by default, equals three.

It remains important to ensure that the LLM refinement (if enabled in AMIE settings) is not repeated in this second run of the mining process because indefinite refinement of rules is infeasible. This addresses the issue of to which extent the LLM can discover new rules by only allowing the LLM to refine rules once. The result of the second run of the mining process, i.e., the *refined output set*, is then appended to the original mining result created by the first run of the mining process. Section 6.4 describes the detailed implementation of the LLM refinement task and illustrates this integration in more detail. Algorithm 5.5 describes the detailed steps for integrating the refinement tasks within AMIE, resulting in newly discovered rules by the LLM.

Algorithm 5.6 describes the main mining process, with a focus on how the mining process is rerun on the newly discovered rules by the LLM refinement task. This includes showing the integration of Algorithms 5.1 and 5.3 of the LLM initial queue task and the evaluation task, respectively. The first run of the Algorithm 5.6 initializes the queue (line 4) using the procedure described in Section 5.2, where the *init* input of this algorithm is set to *true*. The second run initializes the queue (line 7) using the rules discovered by the LLM refinement task, where the *init* input of this algorithm is set to *false*. AMIE evaluation and refinement are then applied (line 9), resulting in mined and discarded rules. The algorithm ensures that the refinement task is not repeated using the stop condition in line 13, where the current mined rules of the second run are returned to be added to the rules mined in the first run.

Algorithm 5.5: Integrating LLM Refinement Task within Algorithm

Input: A list of rules outputted in the first iteration of the mining process
Output: A list of rule objects

```
1  newDiscoveredRules = [];
2  foreach batch in batches do
3  |  Build the prompt using batch and configure the request;
4  |  jsonResponse = generate response by LLM;
5  |  currentRules = parseLLMGeneratedRules(jsonResponse);
6  |  newDiscoveredRules.addAll(currentRules)
7  end
8  refinedRules ← mine(false, newDiscoveredRules);
9  foreach refinedRule in refinedRules do
10 |  if refinedRule ∉ result then
11 |  |  Add refinedRule to result;
12 |  end
13 end
```

Algorithm 5.6: mine(): Mining Process

Input: init, refinedRules, kb
Output: result

```
1  result ← {};
2  discarded ← {};
3  if init then
4  |  seedRules ← LLMInitializeQueueTask(kb.relations) (Algorithm 5.1);
5  end
6  else
7  |  seedRules ← refinedRules;
8  end
9  AMIE mining process: mined rules in result and others in discarded
10 if Evaluation is enabled then
11 |  result ← LLMEvaluationTask(discarded) (Algorithm 5.3);
12 end
13 if !init then
14 |  return result;
15 end
16 if Refinement is enabled then
17 |  result ← LLMRefinementTask(result) (Algorithm 5.5);
18 end
19 return result;
```

5.5 Results and Analysis Approach

Various experiments were conducted to evaluate the approach and its implementation, specifically to assess each LLM task's effectiveness on the mining process, its dependency on other LLM tasks, and its contribution to mining rules. Specific considerations and assumptions were made before analyzing the results of the experiments. This section defines the approach and objectives of conducting the experiments and highlights some assumptions for objectively analyzing the results.

5.5.1 Objectives of the Experiments

The purpose of the experiments is to evaluate the following objectives, which relate to the research questions defined in Section 1.2.

1. **Objective 1:** the impact of the individual LLM tasks on the mining result in terms of quantity and effectiveness
2. **Objective 2:** the impact of the individual LLM tasks on the mining process, such as runtime, resource usage, and cost
3. **Objective 3:** the impact of the individual LLM tasks on the result of the other LLM tasks
4. **Objective 4:** the quality of the rules outputted due to one or more LLM tasks
5. **Objective 5:** the effectiveness of the LLM refinement tasks in discovering new rules from outside the KB in the same domains

Different settings were defined to run the experiments to fulfill these objectives, as shown in Sections 6.6.2 and 6.5.

5.5.2 Considerations and Assumptions

The approach to objectively analyze the results of the experiments aims to comprehensively test all possible contributions the LLM can make to AMIE. Additionally, considering that the LLM might generate slightly different responses for the same prompt due to the so-called LLM hallucination phenomenon [HYM+25], the approach restricts the experiments based on specific assumptions and considerations.

Assumptions

To ensure the analysis of the experiment results is consistent across all results, it is assumed that AMIE finds the maximum number of rules that can be mined from a KB using the default pruning strategies thresholds. Therefore, the experiments are all executed using the default settings of AMIE3.5.

The analysis of the results is based on three assumptions based on the description of AMIE3.5 in Section 4.

- Rules outputted by AMIE evaluation, i.e., fulfill the default pruning strategy criteria (see Section 4.2.3), and are not impacted by LLM evaluation task, are assumed correct and have a valid context.
- No more rules that pass AMIE evaluation, i.e., fulfill pruning strategy criteria, can be mined without using the LLM evaluation task.
- Running AMIE without any LLM task is the ground truth baseline on which the analysis of the experiments is based (see Section 7.1).

Considerations

The existence of a rule in the output can originate from the LLM if it is affected by one or more LLM tasks, either directly or indirectly. A rule directly originating from LLM is generated by either the LLM initial queue task or the LLM refinement task. A rule indirectly originates from the LLM if AMIE refinement operators refine it and its context is evaluated to *true*. Additionally, rules might have originated from AMIE without the intervention of LLM. Highlighting how each rule is derived is crucial to analyze the results. Therefore, each rule has three flags highlighting its origin, as described in Section 6.5. Table 5.1 shows what is considered an origin depending on the flag of a rule.

Rule Flag	**LLM originated?**	**Description**
LLM Created Rule	yes, directly	Rules generated by the LLM initial queue task
LLM Evaluated Rule	yes, directly and indirectly	Rules whose context is evaluated to *true* by the LLM evaluation task
LLM Refined Rule	yes, directly	Rules generated by the LLM refinement task based on the rules outputted
Neither	no, AMIE originated	Rules that has all LLM flags set to ***false***

Table 5.1: Rule sources

In addition to identifying the origin of a rule, the dependency between the LLM tasks is considered crucial to identify how they affect each other. For example, suppose a rule in the mining result originates from the LLM initial queue task and is evaluated by LLM, i.e., both rule flags are set to *true*. In that case, it can be deduced that this rule did

not pass AMIE evaluation, and therefore, it is found in the output set only because the LLM evaluated its context to *true*. Such rules usually have low-quality metrics because AMIE could not find enough evidence for them in the KB, but they are considered valid because they infer a valid context. Additionally, a rule can't be created simultaneously by the LLM initial queue task and the LLM refinement task. Still, the initial queue task affects the refinement task, while the vice versa does not apply. Finally, a rule evaluated to *false* by the LLM evaluation task is never outputted because this task is considered the final step of deciding whether to output a rule.

To measure the effect of the individual LLM task on the runtime of the mining process, Section 6.5 mentions that the time spent on each LLM task is captured. This is because it is infeasible to use the same hardware that ran the mining process in the baseline results of AMIE3.5 [LGS20]. Therefore, the approach mentioned for determining the LLM time is crucial to objectively measuring the effect of the LLM tasks on the runtime.

CHAPTER 6

Implementation

To implement the approach described in Section 5.1, the AMIE3.5 base code must be adjusted to integrate the LLM tasks. Specifically, the LLM Gemini from Google is chosen for this work, which guides the implementation. This section explains the detailed implementation of the approach and the adjustments to the AMIE3.5 algorithm. The code is available on `https://github.com/dbai-tuw/AMIE-LLM`.

6.1 Gemini Client

Most LLMs provide an API used by third-party applications and developers to integrate communication with the LLMs in their code base from the client side. The first step for implementing our approach is implementing a Gemini client responsible for communicating with Gemini through its APIs. This section details how the communication with Gemini API is realized and describes other functionalities the Gemini client offers.

6.1.1 Communicating with Gemini API

Vertex AI is a unified platform Google Cloud provides to perform machine learning operations, including training, validating, testing, and deploying machine learning models. This work uses the Vertex AI library for Java to implement a Gemini client integrated within AMIE to facilitate communication with Gemini APIs. Vertex AI provides different configurations for choosing a specific Gemini model, a Google Cloud project used for billing and other Google Cloud operations, and for setting up Gemini parameters such as controlled content generation.

In addition to the basic request and response behavior provided by Vertex AI, the library also includes the functionality to implement chatting behavior with Gemini. Additionally, Vertex AI can stream responses instead of the classic synchronous communication that blocks the thread. For the approach presented in Chapter 5, it is not necessary to

implement such behaviors, as basic synchronous communication is sufficient in our use case.

The LLM is designed to process input prompts and provide textual output in an indeterministic way, i.e., it might output different texts for the same input prompt. To overcome this, many LLMs can control the output, providing a unified output format. This work uses JSON format to facilitate communication with the LLM.

A JSON schema is designed for each of the three LLM tasks, which is sent to Gemini using the GenerationConfig interface provided by the Vertex AI library. This interface allows the definition of the JSON schema in a variable sent to the LLM before as a parameter.

The corresponding sections in this chapter show the JSON schema for each LLM task.

6.1.2 Throttling Requests

LLM services provided through APIs usually are part of larger platforms that manage billing, access, authorization, and other aspects. The use of LLM services comes with limits defined in tiers that can be free, pay-as-you-go, or with various paid offers, i.e., subscriptions. Gemini APIs are offered through Google Cloud Platform (GCP), which requires setting up a project and a billing account to manage Gemini usage through their APIs. With the pay-as-you-go service, charges begin once the free tier limits are exceeded, automatically switching between the two tiers as needed. For instance, the free tier of the Gemini 1.5-Flash model offers the first 15 requests per minute for free, while every subsequent request starting from the 16th request is charged until the minute has passed. Afterward, the requests counter will be reset, and the first 15 requests of the next minute will be free again.

This work aims to minimize the costs of using the LLM in AMIE. Considering the LLM tasks, it can be expected the limits of the free tier might be exceeded when enabling LLM evaluation and refinement tasks. Therefore, to avoid costs, the Gemini Client implements Algorithm 10.1 that controls the requests sent to the LLM by throttling requests. It tracks the number of requests and pauses further requests once the free requests-per-minute limit is reached, resuming only after the required waiting period has elapsed since the first request.

6.1.3 Batching

Like throttling requests, Gemini offers a limit on output tokens when responding to a prompt. For the evaluation task, all rules discarded by AMIE evaluation and have more than three atoms are sent to Gemini for contextual evaluation. The number of rules to be evaluated depends on the inputted KB, so a general approach should be implemented to avoid Gemini cutting down responses if the number of tokens in the generated content is larger than the output tokens limit. For example, if 1000 rules are sent for evaluation, and the length of the JSON string response of evaluating 1000 rules is larger than the

output tokens limit, we would have the case that not all rules are evaluated, leading to missing information and inconsistent handling of the result. If the mentioned case occurs, no exception will be thrown when using the GenerationConfig interface. Still, a cut-down and malformed JSON string would be returned if the GenerationConfig interface is unused.

The rules are sent in batches using multiple requests to avoid this behavior. The batch number depends on the LLM task and the specific Gemini model used. The next sections describe the batching of each LLM task, if applicable.

6.2 Implementing LLM Initial Queue Task

Implementing the initial queue task requires sending a prompt to the LLM, parsing the LLM response, and defining the procedure to use the generated rules suggested by the LLM, i.e., integrating them into AMIE's initial queue. Sending the prompt happens by simply using the `generateContent()` method defined by Vertex AI library and used in Gemini client. The following describes the implementation of building the prompt and parsing the LLM response(s) in the initial queue task, resulting in rule objects that are used according to Algorithm 5.2.

In case request repetition is enabled, i.e., the corresponding environment variable is set to a value $N > 1$ (see Section 6.6.2), N identical requests will be sent to the LLM in a for-loop, as shown in Algorithm 5.1. Within the loop, the LLM response is parsed into a list of rule objects, followed by the selection strategy that adds unique or duplicated rule objects to a list. After N responses have been parsed, the final resulting list of rule objects is used based on the initial queue initialization mode, as described in Section 5.2.

6.2.1 Prompt

The prompt template for the initial queue task is as follows:

Prompt

You are a logical rule generator for an AMIE miner. Your task is to generate logical rules in the following format:

Template:
`?a relationX ?b ?c relationY ?a => ?c relationZ ?a`

Rules for Variables and Relations:

1. Variables must follow the format defined by the regex `^\?[a-z][0-9]$`. Examples: `?a`, `?b1`, `?c2`.
2. Each relation (`relationX`, `relationY`, `relationZ`) must represent a

logical or meaningful connection between the variables. Examples of relations: `worksAt`, `livesIn`, `hasFriend`.

Guidelines for Rules:

1. Variables must follow the format defined by the regex `^\?[a-z][0-9]$`. Examples: `?a`, `?b1`, `?c2`.
2. Each relation (`relationX`, `relationY`, `relationZ`) must represent a logical or meaningful connection between the variables.
3. Avoid repetitive patterns and explore deeper logical connections:
   - For example, try generating rules that involve unique combinations of relations.
   - Include rules that imply causality, temporality, or hierarchical relationships.
4. Use multi-step logical reasoning wherever possible. For example:
   - If `?a influences ?b` and `?b isLeaderOf ?c`, then `?a might influence ?c`.
5. Rules must only use relations from the provided list.

Examples of Correct Rules:

1. `?a worksAt ?b ?c graduatedFrom ?b => ?c worksAt ?b`
2. `?a p0:directed ?b ?b p0:hasInternalWikipediaLinkTo ?a => ?a p0:created ?b`
3. `?b hasOfficialLanguage ?c ?a livesIn ?b => ?a speaks ?c`
4. `?c hasOrigin ?d ?a imports ?d => ?c dealsWith ?a`
5. `?l isLeaderOf ?c ?b livesIn ?c => ?b hasLeader ?l`

Task:
Generate a set of logical rules following the format and guidelines above. Ensure diversity in logical connections, avoid repetitive patterns, and provide creative and meaningful rules.
List of Relations:
// list of relation provided here, each relation in a line

Section 4.2 explained how AMIE keeps multiple indexes in memory. This includes a basic

list of integers that represent all KB relations. As the prompt above requires appending a list of relations that the LLM understands so it can build logically sound rules, the list of integers representing all KB relations is not suitable because its contents are simple integers. Therefore, the function `unmap()` is provided by AMIE to obtain the string representation of each relation in the list. The result of this function is the original string of each KB relation, which is appended to the prompt and eventually sent to Gemini to generate rules. To send the prompt to Gemini, the `generateContent(()` method of Vertex AI is used, which blocks the current thread until the LLM finishes generating its response and sends it back in JSON format as provided in the JSON schema.

6.2.2 Parse Gemini Response

The controlled content generation works by defining a JSON schema passed to Gemini as a request parameter, which Gemini uses to generate the content based on the given schema. For the initial queue task, the response can contain a simple list of generated rules in string format. Therefore, the JSON schema passed to the LLM in this task looks as follows:

```
{
    "rules": ["rule1", "rule1", "ruleN"],
}
```

For each prompt, Gemini returns a JSON string with that schema, which is then parsed using Java code to get a list of suggested rule strings. AMIE contains an implementation that parses rule strings given in a defined format and builds Rule objects used in the mining process. The Rule object contains many fields, such as rule metrics, rule ancestors, and other fields required by the evaluation and refinement phases. As rule ancestors are irrelevant in the case of LLM-generated rules, the focus is on setting up the initial rule metrics, such as support, head coverage, and PCA confidence.

6.3 Implementing LLM Evaluation Task

All rules with more than three atoms discarded by AMIE evaluation are included for this task and then sent to Gemini for contextual assessment. As mentioned in Section 5.1, the rules to be evaluated are aggregated and sent together to avoid runtime overhead caused by Gemini response time, throttling requests, and parsing processing time. Therefore, in the main loop, when AMIE decides that a rule is unsuitable to be added to the result set and contains three atoms, it will be added to a list. As AMIE3 uses multi-threading to boost performance, this list containing all discarded rules should be thread-safe. After the main loop, the LLM evaluation procedure begins by processing the list containing all discarded rules.

As explained in Section 6.1.3, Gemini has its (free) limits, so batching addresses the issue of output tokens count that can be easily exceeded with many usages in our

implementation, such as the evaluation task. Not all discarded rules can be sent to Gemini at once to evaluate their contextual information because it might cut down its response, i.e., evaluate only a subset of the sent rules without throwing an exception, making the result incomplete. Therefore, the discarded set of rules that must be evaluated is batched. The batch size depends on the Gemini model that defines the limits, which is 150 rules per batch when using Gemini 1.5-Flash and 20 rules per batch when using Gemini 1.5-Pro. See Chapter 7 for more detailed experiments.

6.3.1 Prompt

The prompt template for sending out N rules is as follows:

Prompt

In the area of rule mining knowledge engineering, logical horn rules are rules in the following format:
Template:
`?a relationX ?b ?c relationY ?a => ?c relationZ ?a`

Example:
`?b hasOfficialLanguage ?c ?a livesIn ?b => ?a speaks ?c`

Explanation of the example:
If person ?a has official language ?c, and ?a lives in place ?b, then person ?a speaks official language ?c

The context of that example is correct. Similarly, evaluate the context of the logical rules below to true or false.

Logical rules:

6.3.2 Parse Gemini Response

Gemini client uses controlled generation to control the output in JSON format. Therefore, just like the initial queue task, this task requires defining a JSON schema to be sent as a parameter to Gemini with each request. The JSON schema is then used to parse the output of Gemini, in this task the contextual evaluation result of each rule is sent. The JSON schema for the evaluation task is as follows:

```
{
    "rules": [
        {"rule": "ruleString", "context": "evalResult"}
    ]
}
```

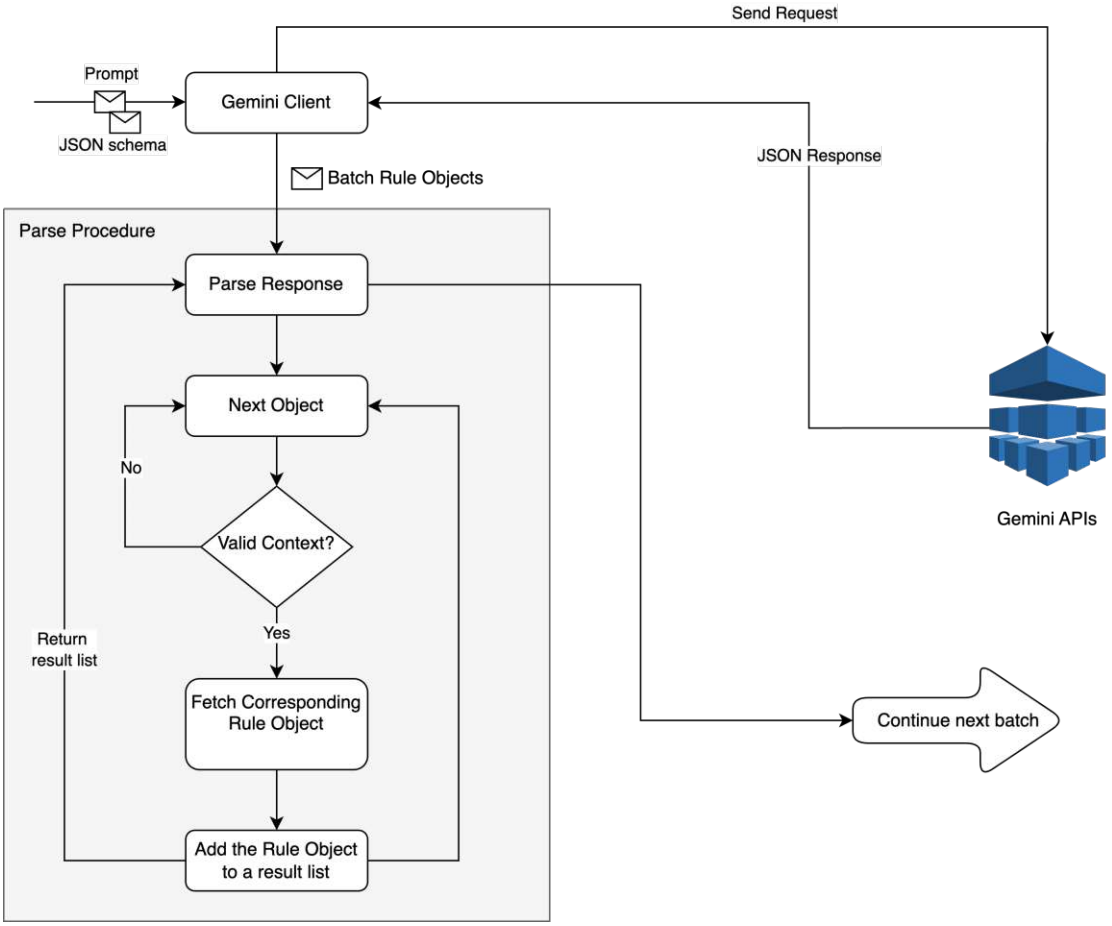

Figure 6.1: Parsing procedure of evaluation task.

This schema allows for easy parsing of the response. Gemini will put each rule sent in the prompt in an object alongside its contextual evaluation result (boolean). When parsing the Gemini response, a JSON string, the objects in the list under the 'rules' key are processed. For each object, if the context is *true*, the corresponding rule is identified within the list containing AMIE rule objects by matching it to the rule string in the current object.

Figure 6.1 illustrates the evaluation task and outlines the procedure of parsing the Gemini JSON response. In the beginning, the Gemini Client is used to make a request to Gemini, providing it with the prompt and JSON schema for controlled output generation. The parsing procedure is provided with the list of rule objects that are required to find what rule objects correspond to which evaluated rule string. Using the rule objects is essential for AMIE, which requires all information about a rule to be included within each object.

After the Gemini response is parsed and all to-true evaluated rule objects in the current batch are identified, they are added to the result set of AMIE. The same procedure

is repeated until all batches have been processed, i.e., all discarded rules have been contextually evaluated by Gemini, thereby concluding the LLM evaluation task.

6.4 Implementing LLM Refinement Task

As complementary to the default AMIE refinement procedure, the LLM refinement task aims to find rules with atoms from outside the KB in the same domain as the rules mined by AMIE. Gemini will be provided with a list of rules and is requested to generate new rules with similar context in the same domain of each provided rule by incorporating its domain knowledge. Conclusively, the LLM refinement task is run at the end of the mining process, after the main loop terminates, and after the LLM evaluation task concludes.

The implementation includes batching the list of outputted rules that AMIE created. Similar to the evaluation task, every batch is appended to the prompt template (see below), resulting in a prompt input sent to the LLM alongside a GenerationConfig object that defines the controlled output generation response using JSON schema. The JSON schema for this task is the same as for the initial queue task shown in Section 6.2. The implementation of the refinement task follows Algorithms 5.5 and 5.6, which are illustrated in Figure 6.2.

6.4.1 Prompt

The prompt template for the refinement task is as follows:

Prompt

You are a logical rule generator for an AMIE miner. Your task is to generate logical rules in the following format:

Template:
`?a relationX ?b ?c relationY ?a => ?c relationZ ?a`

Rules for Variables and Relations:

1. Variables must follow the format defined by the regex `^\?[a-z][0-9]$`. Examples: `?a`, `?b1`, `?c2`.
2. Each relation (`relationX`, `relationY`, `relationZ`) must represent a logical or meaningful connection between the variables. Examples of relations: `worksAt`, `livesIn`, `hasFriend`.

Guidelines for Rules:

1. Variables must follow the format defined by the regex `^\?[a-z][0-9]$`. Examples: `?a`, `?b1`, `?c2`.

2. Each relation (`relationX`, `relationY`, `relationZ`) must represent a logical or meaningful connection between the variables.
3. Avoid repetitive patterns and explore deeper logical connections:
   - For example, try generating rules that involve unique combinations of relations.
   - Include rules that imply causality, temporality, or hierarchical relationships.
4. Use multi-step logical reasoning wherever possible. For example:
   - If `?a influences ?b` and `?b isLeaderOf ?c`, then `?a might influence ?c`.

Examples of Correct Rules:

1. `?a worksAt ?b ?c graduatedFrom ?b => ?c worksAt ?b`
2. `?a p0:directed ?b ?b p0:hasInternalWikipediaLinkTo ?a => ?a p0:created ?b`
3. `?b hasOfficialLanguage ?c ?a livesIn ?b => ?a speaks ?c`
4. `?c hasOrigin ?d ?a imports ?d => ?c dealsWith ?a`
5. `?l isLeaderOf ?c ?b livesIn ?c => ?b hasLeader ?l`

Task:
Below is a list of rules outputted by AMIE. Use the context of these rules to generate as many logical rules as possible by incorporating domain knowledge. Make sure to follow the format and guidelines above. Ensure diversity in logical connections, avoid repetitive patterns, and provide creative and meaningful rules.
List of rules outputted by AMIE:
// list of relation provided here, each relation in a line

6.5 Tracking and Statistics

Implementing the LLM tasks focuses on building prompts, controlling the communication with Gemini, and parsing its response. When integrating these tasks within AMIE, described in Section 6.6, it is essential to distinguish the rules originating from the LLM, as described in Section 5.5.2. Specifically, a rule can either be created, evaluated, or refined by the LLM or outputted without any intervention from the LLM, i.e., AMIE outputted rules. This work defines flags for each rule in the Rule object to track LLM

tasks, as shown in Table 6.1.

Flag	Description
LLM-Created	Rule was generated by LLM in the Initial Queue Task
LLM-Evaluated	Rule was evaluated by LLM in the Evaluation Task
LLM-Refined	Rule was refined by LLM in the Refinement Task

Table 6.1: LLM Flags and their descriptions.

Although the rules generated in the initial queue task and refinement task are both generated by the LLM, we do not set the LLM Created flag to `true` for the rule generated in the LLM refinement Task to clearly distinguish the initial Queue Task from the refinement Task. This means that the LLM-Created flag and the LLM-Refined flag cannot be set to `true` for the same rule. The LLM-Evaluated flag can be set with either of the other two for the same rule.

In addition to defining flags, the time invested in performing each LLM task is tracked to measure the effect of LLM integration on the overall runtime. Within each task, we capture the timestamp just before building the prompt for the LLM. Once a response is returned, the captured timestamp is subtracted from the current timestamp to get the actual time invested by the LLM.

Additionally, this work gathers statistics and counts during the entire run of AMIE to provide valuable insights on all aspects of the LLM tasks and to measure the benefits and drawbacks of each task. The statistics gathered are shown in Table 6.2.

Count	Description
LLM Suggested Rules	rules generated by initial queue task
LLM evaluation calls	LLM requests made in evaluation task
LLM refinement calls	LLM requests made in refinement task
Rules to be LLM evaluated	all rules sent to LLM for evaluation
Rules to be LLM refined	all rules sent to LLM for refinement
Rules LLM refined	rules generated by refinement task
Rules with valid context	rules evaluated to true by evaluation task
Rules with invalid context	rules evaluated to false by evaluation task
Outputted LLM suggested rules	rules generated by initial queue task and got outputted
Outputted rules with valid context	rules that got outputted due to their contextual validity
Outputted LLM refined rules	rules generated by refinement task and got outputted

Table 6.2: Statistics and Counts

6.6 Integration of LLM Tasks into AMIE Algorithm

The LLM tasks require integrating them within the AMIE algorithm by modifying them to provide additional information, modify the logical flow, and perform control actions for the LLM tasks. Additionally, this work aims to create a clear separation between AMIE and LLM tasks. On the one hand, to have a good degree of flexibility in the mining process and, on the other hand, to ensure the original AMIE mining algorithm remains a standalone entity in the project. This section shows how the integration of each LLM task into AMIE is implemented and how the separation between AMIE's mining process and the LLM tasks is realized.

6.6.1 Integration into AMIE

Each LLM task requires input from AMIE and provides an output used in AMIE. The following table summarizes the information required for each task and the responses to be used.

LLM Task	**Input**	**Output**
Initial Queue Task	KB relations	suggested rule objects
Evaluation Task	discarded rule objects	contextually valid rules
Refinement Task	outputted rule objects	refined rule objects

Table 6.3: LLM Tasks and their inputs/outputs (lists)

The inputs and outputs of the LLM tasks make it easier to separate the LLM tasks from the AMIE mining algorithm, thereby providing high control and flexibility. Figure 6.2 illustrates the procedure of AMIE with all LLM tasks integrated. All LLM tasks can be activated separately using control parameters, as described in the following section.

To provide the list of KB relations to the initial queue task, the list of KB relations is passed as a parameter as a parameter, which requires no code modifications. As for the initial queue task output, the AMIE code is modified to implement the initial queue initialization modes, as described in Algorithm 5.2. Hence, the initial queue is reassigned with the initial queue task output in case the mode is ***ONLY_LLM***, and is appended with the output in case the mode is ***LLM_AND_AMIE***. Otherwise, it is the third mode ***ONLY_AMIE***, which does not use the LLM suggested rules in the initial queue, and hence, no call to Gemini is made, and the LLM initial queue task is considered deactivated.

More complex and extensive code modifications must be made to integrate the evaluation task, as it requires a list of all rules discarded by AMIE. Therefore, AMIE code is modified to include (1) the implementation of a new thread-safe list of rules that contains AMIE discarded rules and (2) the implementation of the two modes for evaluation, ***SINGLE*** mode and ***AGGREGATED*** mode, and (3) the implementation of the evaluation procedure that batches the discarded rules, throttles requests to Gemini, and parses the evaluation result as explained in Section 5.3.

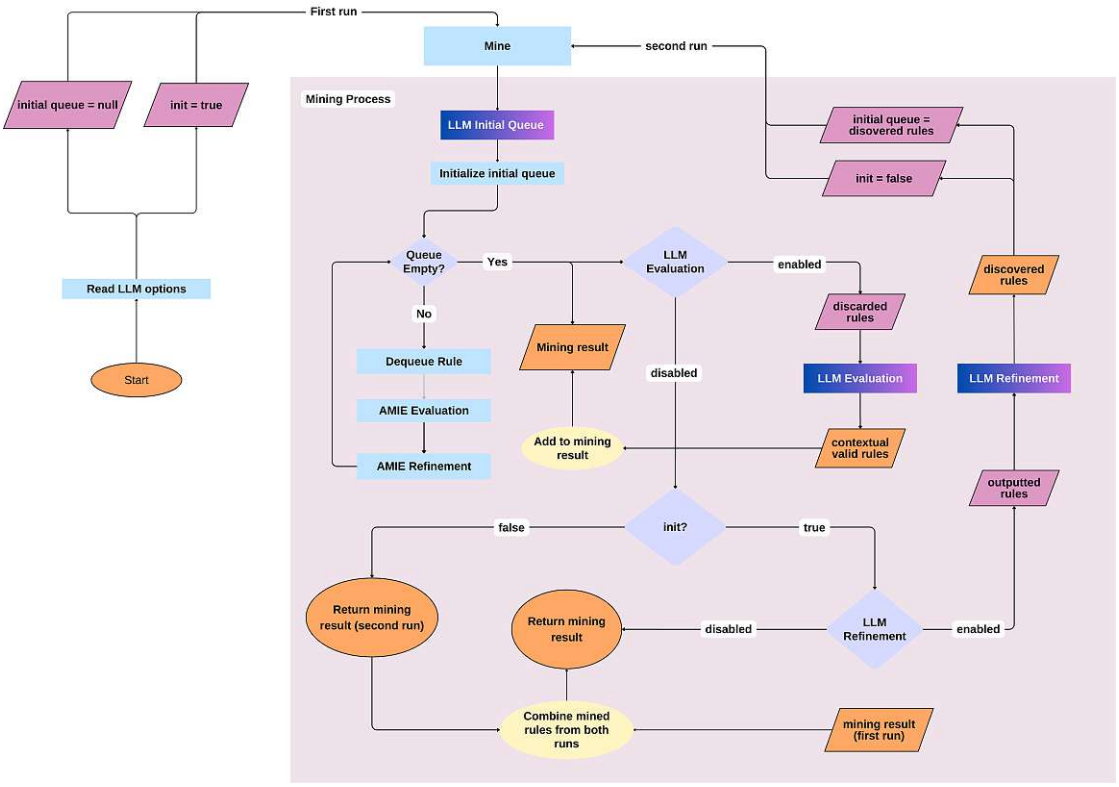

Figure 6.2: LLM tasks integration within AMIE.

As for the input defined for the refinement task, AMIE only has to pass the rules corresponding to the mining result as a parameter, similar to the initial task input. However, the refinement task output, containing refined and newly discovered rules, must be evaluated by rerunning the mining process to perform AMIE and LLM evaluations. Hence, the main mining method is modified to consume a list of rule objects as a parameter and a boolean to indicate that it is the second run of the mining process, following the description of Algorithm 5.6. The LLM refinement procedure is not repeated in the second run, and the initial queue contains only the rules from the method input parameter. When the second run of the mining process is finished, the resulting rules are appended to the original one, providing the final result.

6.6.2 Control Parameters

This work defines control parameters loaded at AMIE's startup from environment variables to achieve high control and flexibility when running the mining procedure with the integrated LLM tasks. These variables activate and deactivate single LLM tasks and define their modes and options. Table 6.4 shows all available control parameters passed as environment variables.

env variable	**description**	**corresponding LLM task**
LLM_INIT_MODE	initial queue initialization mode	initial queue task
LLM_INIT_SAMPLES	request repetition number	initial queue task
LLM_SELECTION_MODE	selection strategy of suggested rules	initial queue task
LLM_EVAL_ENABLE	enable or disable LLM evaluation	evaluation task
LLM_EVAL_MODE	evaluation mode	evaluation task
LLM_REFINEMENT_ENABLE	enable or disable LLM refinement	refinement task
OUT_FILE	write AMIE output logs to a file	-

Table 6.4: Control Parameters for running AMIE with LLM Tasks

6.6.3 Logs and Outputs

During the execution of the AMIE system, detailed log files are meticulously maintained to facilitate an in-depth analysis of the response quality across all LLM tasks. For instance, the response of the initial queue task is saved into a file with a clear naming convention. The same applies to the evaluation task and refinement task. Additionally, meaningful log messages are found across the mining process to log various information relevant to measuring the quality of the project and to keep track of the overall mining process.

At the end of the mining process, and when all LLM tasks have been successfully finished, a list of rules is outputted alongside its quality metrics. In addition to this, the flags explained in Section 6.5 are outputted for each rule to show the rules affected by the LLM. Finally, the statistics relevant to LLM tasks gathered across the mining process are outputted and used to analyze and measure the work quality.

CHAPTER 7

Experiments and Results

This chapter describes the experiments conducted to fulfill this work's objectives, mentioned in section 5.5.1, and their results. Chapter 8 discusses the results and the fulfillment of the objectives and research questions.

7.1 Experimental Setup

This section describes the knowledge base to be mined, the hardware used to run the experiments, the LLM chosen, and the different experiments conducted.

7.1.1 Knowledge Base and Configuration

The experiment results are analyzed using a reference, i.e., a baseline, to objectively measure the improvement made by integrating LLM in AMIE. This implies that the knowledge base used in the experiments should contain the same facts as those in the knowledge base used in the mining process that produces the baseline results. This can be done by simply using the same knowledge base.

The baseline used as a reference for analyzing the results is the experiment conducted in [LGS20] on the knowledge base YAGO2 [HSBW13]. Consequently, the knowledge base used to run all the experiments in this work is YAGO2. The choice of the baseline is because the integration of the LLM is done on AMIE's latest version, AMIE3.5, of which an improvement should be measured. Additionally, the knowledge base choice of YAGO2 is due to its high number of facts and the ability to use it efficiently, as all facts are included in one file in turtle format, which is ideal for running AMIE.

Yago2 is a large knowledge base containing facts in turtle format [BP25], which formats facts as RDF triple in subject-predicate-object, as explained in section 2.1.4. Yago2 includes general knowledge about people, cities, countries, movies, and organizations and includes around 447 million facts and 10 million entities [HSBW13].

Configurations

The hardware used to run the experiments in this work is shown in table 7.1.

Component	Specification
OS	Debian GNU/Linux 11 (bullseye)
CPU	Intel(R) Xeon(R) Silver 4314 @ 2.40GHz
CPU Cores	16
RAM	1007 GB
Java Version	OpenJDK 19

Table 7.1: System Specifications

7.1.2 Baseline

The baseline results used when analyzing the results of all experiments to measure the improvements and drawbacks of LLM integration in AMIE are the results obtained from running AMIE in default settings without enabling any LLM task, i.e., pure AMIE mining algorithm, as described in chapter 4. This ensures an objective and accurate analysis of the impact of LLM on the mining result. AMIE originally loads 110 million facts from Yago2, with 96 relations (including 5 RDFS relations defined in Yago2). Additionally, AMIE needs 19 minutes and 11 seconds to load Yago2 in the integer-based in-memory database 4.3.1, using 180 GB of RAM. The mining process finishes in 1 hour and 32 minutes, resulting in 124 mined rules, with a total runtime of 1 hour and 51 minutes. Table 7.2 shows the baseline for all experiments for better visualization.

Parameter	Value
Number of Facts	110 million
Number of Relations	96 (including 5 RDFS relations)
RAM Used for In-Memory Database	180 GB
Total number of Threads Used	64 Threads
Yago2 Loading Time	19 minutes 11 seconds
Mining Time	1 hour 32 minutes
Total Runtime	1 hour 51 minutes
Mined Rules	124

Table 7.2: Baseline on Yago2 using Default AMIE Settings

It is worth mentioning that the total RAM AMIE uses until the exit, i.e., until all 124 rules are mined, is around 280 GB, using multi-threading.

7.1.3 LLM and Prompting

As mentioned in chapter 5, Gemini is used in this work as the LLM responsible for generating content corresponding to each LLM task. Gemini has various models, each differently optimized and fine-tuned to specific tasks, and each cost differently based on usage. Additionally, the model's reasoning ability is crucial for the quality of the responses, as some models prefer faster content generation over reasoning, and others favor reasoning and generation content quality over the length of responses, etc.

To address these variations, the experiments in this work are made using two Gemini models: Gemini 1.5-Flash and Gemini 1.5-Pro. The former is fast at generating responses with less reasoning power, while the latter prefers better reasoning over performance.

7.1.4 Experimental Environment

This work runs the implemented code in a containerized environment using Docker to benefit from the portability of container images. Therefore, the code is compiled and packaged into a jar file, which is transferred to the computer that runs the experiments where the knowledge base is saved. The settings described in Section 6.6.2 are passed to the modified AMIE using an environment variable file (`.env` file), which is loaded at the startup of AMIE. The Dockerfile used executes multiple commands required to set the environment in which the code runs, including:

1. setting up container image to use Java 19
2. setting up local Google Cloud Platform using gCloud CLI to enable Gemini API calls communication from the Docker container through the network
3. copying the current jar file to be executed to the container
4. copying the current `.env` file required by the code

Every experiment was started using a shell script, which builds a Docker container using the Dockerfile, where the container name, container port, and knowledge base file are configured as an input to the container through the shell script, ensuring flexibility and enabling running multiple experiments simultaneously.

7.1.5 Experiments Considerations

The approach and implementation presented in Chapters 5 and 6 describe the different settings introduced to AMIE to control the LLM integration and to ensure their independence from AMIE's main algorithm. The proposed settings include parameters to enable/disable individual LLM tasks and specify the selection mode and request repetition number for the initial queue task, explained in Section 6.2.

Based on the proposed settings, 12 experiments can be conducted without changing the request repetition number, i.e., locking it to 1 (no request repetition). This is the number of experiments combinatorially computed, considering evaluation and refinement tasks permutation (4 experiments) and three modes for the initial queue task, including not enabling it. Additionally, five experiments were performed to measure the effect of the request repetition concept, which used different selection strategies in the initial queue task without combinatorial creating experiments.

The experiments are designed with a specific aim at one or more objectives. For instance, the experiments made to measure the quality of the request repetition concept include running in the initial queue mode ***ONLY_LLM***, with selection mode ***ALL***. It remains important to test how the LLM tasks are connected, i.e., the impact of each on the other. Therefore, 12 experiments are necessary to observe and highlight the effects.

7.2 Experiments

The objectives in Section 5.5.1 are essential to accurately determine the effectiveness of Large Language Models in rule mining in general. The results analysis will depend on measuring the extent to which each objective is met by the LLM integration in AMIE, which requires comprehensive experiments to obtain meaningful results. This section will describe the different experiments made in this work.

7.2.1 LLM Tasks Impacts

Objective 3 focuses on analyzing the impact of each LLM task on the others. This is achieved through experiments that vary the order of LLM tasks, including the three modes of the LLM's initial queue task. In total, 12 distinct experiments are conducted, excluding those that assess the effect of request repetition. Table 7.3 shows the experiments performed and their objectives.

The individual efficiency of each LLM task is measured, including its contribution to the mining result (objectives 1 and 2), and the quality of each rule (objective 4) can be measured. Additionally, in the experiments where the LLM refinement task is enabled, the ability of the LLM to discover new rules from outside the KB in the same domain is tested.

7.2.2 Request Repetition

To measure the effect of request repetition, the repetition number and the selection mode parameters in the environment variables file (`.env`) are set. The following tables list the experiments to test the effectiveness of request repetition on the initial queue of AMIE under the LLM initial queue task. These experiments are not repeated for each of the Gemini models. For Gemini 1.5-Pro, the experiments in Table 7.5 are different and focus more on testing the rule quality, which results are discussed in detail in Section 8.1.

ID	Init Queue Mode	LLM Task(s)	Objectives
1F, 1P	ONLY_AMIE	None	(Baseline)
2F, 2P	ONLY_AMIE	Evaluation	1, 2, 4
3F, 3P	ONLY_AMIE	Refinement	1, 2, 4, 5
4F, 4P	ONLY_AMIE	Evaluation + Refinement	1-5
5F, 5P	ONLY_LLM	None	4
6F, 6P	ONLY_LLM	Evaluation	3, 4
7F, 7P	ONLY_LLM	Refinement	3-5
8F, 8P	ONLY_LLM	Evaluation + Refinement	3-5
9F, 9P	LLM_AND_AMIE	None	1, 2, 4
10F, 10P	LLM_AND_AMIE	Evaluation	1-4
11F, 11P	LLM_AND_AMIE	Refinement	1-5
12F, 12P	LLM_AND_AMIE	Evaluation + Refinement	1-5

Table 7.3: Experiments conducted for LLM Tasks impact. The column ID refers to the experiment identification (ID), where each experiment is assigned to two IDs, i.e., repeated using different LLM and prompt, as explained in Section 7.2.3

ID	Init Queue Mode	LLM Task(s)	Selection Mode
13F	ONLY_LLM	None	ALL
14F	ONLY_LLM	Evaluation	DUPLICATE
15F	ONLY_LLM	None	DUPLICATE
16F	LLM_AND_AMIE	Evaluation + Refinement	UNIQUE
17F	LLM_AND_AMIE	Evaluation + Refinement	DUPLICATE

Table 7.4: Experiments using Gemini 1.5-Flash for request repetition concept with repetition number = 8

These experiments aim to test if the request repetition concept could result in more rules generated by the LLM initial queue task in the output set without using LLM evaluation. This directly measures the effectiveness of the LLM initial queue task. Additionally, the selection mode ***DUPLICATE*** might increase the probability of higher quality rules, thereby addressing objective 4, which aims at measuring the quality of rules generated by the LLM. In general, the question of whether the concept of request repetition could statistically improve the mining result will be examined using the results of the experiments of Table 7.4 and 7.5 using Gemini 1.5-Flash and Gemini 1.5-Pro, respectively.

ID	Init Queue Mode	LLM Task(s)	Selection Mode
13P	ONLY_LLM	None	DUPLICATE
14P	ONLY_LLM	Evaluation	DUPLICATE
15P	ONLY_LLM	Evaluation + Refinement	ALL
16P	ONLY_LLM	Evaluation + Refinement	DUPLICATE
17P	ONLY_LLM	Evaluation + Refinement	UNIQUE

Table 7.5: Experiments using Gemini 1.5-Pro for request repetition concept with repetition number = 20

7.2.3 The effect of Gemini Model

As mentioned in Section 7.1.3, Gemini includes different models optimized for specific tasks. To test the effect of the model corresponding to the objectives mentioned earlier, especially objective 4 (rule quality), this work proposes using two different models, i.e., Gemini 1.5-Flash and Gemini 1.5-Pro. Hence, the experiments from Table 7.3 are repeated twice using a different setup, resulting in 24 experiments:

- **Setup F:** Gemini 1.5-Flash, using the prompts in the Appendix 10, referred to as basic prompts.
- **Setup P:** Gemini 1.5-Pro, using the prompts defined and discussed in Section 5.1, referred to as default prompts.

Based on this, the experiments with IDs ending with “F” use setup F, and the experiments with IDs ending with “P” use setup P.

Prompt Effect

The basic prompts presented in Appendix 10 do not result in good results, as shown in Section 7.3. They were the input for the Gemini 1.5-Flash model. To test the effect of changing the prompts, manual requests were made to Gemini 1.5-Flash, which is discussed in Section 7.4

7.3 Results

This section will describe the results of the experiments from Tables 7.3, 7.4 and 7.5. The baseline from Table 7.2 is used to measure the improvements done by each model, specific to the objectives defined in Section 5.5.1.

7.3.1 LLM Initial Queue Task

In this task, Gemini should generate logical rules from the list of KB relations given in the prompt. Tables 7.6 and 7.7 show the results of the experiments with a focus on the LLM initial queue task, including the results of experiments 9 and 10, where the LLM evaluation is enabled, which affects the initial queue directly. As mentioned in Section 5.5.2, the LLM refinement task does not affect the initial queue task. Hence, the results shown in this section do not include the experiments where LLM refinement is enabled. This is discussed later in Section 7.3.3.

ID	# Rules	# Generated	# Outputted	# LLM evaluated
5F	1	87	1	N/A
6F	9	10	9	9
9F	125	164	1	N/A
10F	1603	254	254	254

Table 7.6: Results for LLM initial queue task experiments with Gemini 1.5-Flash

ID	# Rules	# Generated	# Outputted	# LLM evaluated
5P	0	10	0	N/A
6P	1	10	1	1
9P	124	10	0	N/A
10P	464	10	4	4

Table 7.7: Results for LLM initial queue task experiments with Gemini 1.5-Pro

The results in Tables 7.6 and 7.7 show for each experiment the number of total rules outputted, the number of rules generated by the LLM in the initial queue task, and the number of generated rules found in the output. The last column highlights the impact of the LLM evaluation task on the initial queue task by presenting the number of rules evaluated to *true*. Based on the second assumption from Section 5.5.2, these rules were discarded by AMIE and are evaluated by the LLM, so their existence in the mining result is exclusively due to the LLM evaluation task.

7.3.2 LLM Evaluation Task

Unlike the other two LLM tasks, this task evaluates the context of the rules given in the prompt and does not generate any rules. This section highlights the impact of this task on AMIE by showing the results of experiments 2F and 2P, while its impact on the other two tasks is discussed in Sections 7.3.1 and 7.3.3.

The baseline in Table 7.2 shows that AMIE mines 124 and discards 1301 rules. When enabling LLM evaluation in experiments 2F and 2P, these discarded rules are sent to the LLM for contextual evaluation. Tables 7.8 and 7.9 show the results of these experiments.

Metric	2F
# total rules outputted	1347
# discarded rules by AMIE	1301
# evaluated by LLM	1223
# true-evaluated percentage	94%

Table 7.8: Evaluation task results (2F) with Gemini 1.5-Flash

Metric	2P
# total rules outputted	465
# discarded rules by AMIE	1301
# evaluated by LLM	341
# true-evaluated percentage	26%

Table 7.9: Evaluation task results (2P) with Gemini 1.5-Pro

7.3.3 LLM Refinement Task

To test the direct effect of the LLM refinement procedure, experiments 3F and 3P are performed, which use the 124 rules mined by AMIE (baseline) in the first run of the mining process to discover new rules. Tables 7.10 and 7.11 show the results of these experiments.

Metric	3F
# total rules outputted	130
# rules sent for refinement	124
# rules generated by the LLM	120
# rules generated in output	6

Table 7.10: Results for LLM refinement task (3F) with Gemini 1.5-Pro

Metric	3P
# total rules outputted	124
# rules sent for refinement	124
# rules generated by the LLM	46
# rules generated in output	0

Table 7.11: Results for LLM refinement task (3P) with Gemini 1.5-Pro

Task Dependency

When the LLM initial queue task is enabled along with the refinement task, i.e., in experiments 7 and 11, the effect of it on the refinement task can be identified, as shown in Table 7.12 and 7.13, for Gemini 1.5-Flash and Gemini 1.5-Pro, respectively.

The results show that both LLM tasks contribute to the mining results by a limited number of logical rules, although both models generate many rules. The reason is that most of these rules are discarded by AMIE, and consequently, only a few make it to the output. Additionally, the effect of the initial queue task can be seen in the experiment: the more rules generated by the initial queue task are outputted in the first iteration, the more rules can be mined in the second iteration by the refinement task.

ID	# Init Gen.	# Init Outputted	# Ref. Gen.	# Ref. Outputted
7F	89	2	6	1
11F	19	2	112	6

Table 7.12: Results of LLM refinement tasks with the initial queue task enabled. The second and third columns indicate the number of rules generated and outputted by the initial queue task. Similarly, the fourth and fifth columns represent the corresponding values for the refinement task using Gemini 1.5-Flash.

ID	# Init Gen.	# Init Outputted	# Ref. Gen.	# Ref. Outputted
7P	10	1	5	1
11P	14	0	47	0

Table 7.13: Description is the same as Table 7.12, with the difference of using Gemini 1.5-Pro to produce these results.

Enabling the LLM evaluation task in experiment 4 improves the number of outputted rules from the refinement task because the number of rules sent for refinement is higher. Tables 7.14 and 7.15 show the results of experiments 4F and 4P, respectively.

Experiment	**4F**
First Iteration	
AMIE Output	124
Discarded (LLM Eval)	1301
true-evaluated	1123
Total Output	**1247**
Second Iteration	
Sent for Refinement	1247
Rules Generated	910
LLM Evaluated	342
AMIE Evaluated	7
Total Output	349
Total rules	1596

Table 7.14: Results of LLM refinement task with LLM evaluation enabled using Gemini 1.5-Flash

Experiment	**4P**
First Iteration	
AMIE Output	124
Discarded (LLM Eval)	1301
True Evaluated	344
Total Output	468
Second Iteration	
Sent for Refinement	468
Rules Generated	244
LLM Evaluated	103
AMIE Evaluated	0
Total Output	103
Total rules	571

Table 7.15: Results of LLM refinement task with LLM evaluation enabled using Gemini 1.5-Pro

7.3.4 Enabling All Tasks

When enabling all LLM tasks in experiments 8 and 12, the number of rules in the output is expected to be high. Tables 7.16 and 7.17 show the results of experiments 8 and 12 using Gemini 1.5-Flash and Gemini 1.5-Pro, respectively, with a focus on how the LLM initial queue task influences the refinement task, considering that the LLM evaluation is enabled. The impact of the LLM evaluation task on both tasks is shown for the same experiments in Tables 7.18 and 7.19

ID	# Init Gen.	# Init Outputted	# Ref. Gen.	# Ref. Outputted
8	16	16	7	0
12	265	159	1110	463

Table 7.16: Results of all LLM tasks enabled using Gemini 1.5-Flash. Column description is analogous to Table 7.12.

ID	# Init Gen.	# Init Outputted	# Ref. Gen.	# Ref. Outputted
8	10	5	8	2
12	15	3	260	110

Table 7.17: Results of all LLM tasks enabled using Gemini 1.5-Pro. Column description analogue to Table 7.12

Experiment 8 shows that 16 is the total number of rules outputted, including the rules outputted by the LLM initial queue and refinement tasks. Analogous to this, experiment 12 shows that both LLM tasks contribute to mining result by 622 rules. To identify the impact of the LLM evaluation on these results, Tables 7.18 and 7.19 show how many rules are evaluated to *true* for both experiments, outlining the source of the rules.

Experiment	Source	Rules Outputted	Evaluated Rules
8	Initial Queue Task	16	6
8	Refinement Task	7	0
12	Initial Queue Task	265	154
12	Refinement Task	453	453

Table 7.18: Evaluation of rules generated in experiment 8 and overall rule mining performance. The second column indicates whether the rules originated from the initial queue task or the refinement task. The last column shows the number of rules evaluated as *true* by the evaluation task. Using Gemini 1.5-Flash.

Experiment	Source	Rules Outputted	Evaluated Rules
8	Initial Queue Task	10	4
8	Refinement Task	8	2
12	Initial Queue Task	3	3
12	Refinement Task	110	108

Table 7.19: Description the same as Table 7.18, but using Gemini 1.5-Pro to produce these results.

7.3.5 Request Repetition Effect

As mentioned in Section 5.1, the request repetition concept ensures that the LLM generates as many rules as possible and uses all statistically possible relevant relations, which defines three modes for selecting generated rules to be used in the initial queue of AMIE. This section describes the results of experiments 13-17, which were conducted to test the effectiveness of this concept. Tables 7.20 and 7.21 show the results of the experiments related to the request repetition concept for each of the two Gemini models, respectively.

ID	# rules	# Init. Gen.	Init. Selected	LLM Evaluated
13	13	520	520 (all)	N/A
14	42	556	42 (duplicate)	41
15	0	801	80 (duplicate)	N/A
16	2740	743	674 (unique)	615
17	1933	673	97 (duplicate)	97

Table 7.20: Results of request repetition using Gemini 1.5-Flash model

ID	# rules	# Init. Gen.	Init. Selected	LLM Evaluated
13	4	204	35 (duplicated)	N/A
14	18	209	37 (duplicated)	15
15	71	218	218 (all)	65
16	31	200	40 (duplicated)	30
17	49	210	97 (unique)	44

Table 7.21: Results of request repetition using Gemini 1.5-Pro model

7.3.6 Runtime

Table 7.22 and 7.23 shows the detailed mining time spent in each of the three LLM tasks in each of the experiments from Tables 7.3, 7.4, and 7.5, for Gemini 1.5-Flash and Gemini 1.5-Pro, respectively.

ID	Init. Queue	Evaluation	Refinement	LLM time	Total
1	-	-	-	0s	1h 32m
2	-	3m	-	3m	1h 29m
3	-	-	15s	15s	1h 26m
4	-	5m 24s	4m 25s	10m	1h 42m
5	15s	-	-	15s	15s
6	1s	4s	-	4s	5s
7	13s	-	1s	14s	15s
8	30s	4s	1s	35s	36s
9	19s	-	-	19s	1h 26m
10	35s	3m 54s	-	4m 29	1h 29m
11	4s	-	13s	17s	1h 26m
12	29s	6m 8s	3m 19s	9m 54m	1h 44m
13	2m 45s	-	-	2m 45s	3m 28s
14	2m 22s	6s	-	2m 28s	2m 33s
15	1m 59s	-	-	1m 59s	2m 2s
16	1m 37s	8m 35s	6m 44s	16m 56s	1h 50m
17	2m 37s	5m 40s	6m 54s	14m 38s	1h 48m

Table 7.22: Mining Runtime of all experiments using Gemini 1.5-Flash and basic prompt

7.4 Gemini 1.5-Flash with default Prompt

All the results produced by Gemini 1.5-Flash were conducted using the basic prompts defined in the Appendix 10. The default prompts described in Section 5.1 produced better results using Gemini 1.5-Pro. To test the prompt factor on the effectiveness of each LLM task, manual requests were made to Gemini 1.5-Flash to test the improvements made by the default prompt over the basic prompt. This section describes the experimental approach of the manual requests and the results of using this setup.

ID	Init. Queue	Evaluation	Refinement	LLM time	Total
1	-	-	-	0s	1h 32m
2	-	21m 24s	-	21m 24s	2h 7m
3	-	-	31	31s	1h 28m
4	-	24m 22s	2m 39s	27m	2h 26m
5	9s	-	-	9s	10s
6	9s	9s	-	18s	19s
7	9s	-	3s	12s	13s
8	9s	15s	5s	29s	1m 24s
9	9s	-	-	9s	1h 33m
10	9s	10h 57m	-	11m 6s	3h 21m
11	9s	-	21s	30s	2h 45m
12	20s	12m 50s	1m 50s	15m	3h 48m
13	10 mins 22 secs	-	-	10m 22s	10m 22s
14	10 mins 29 secs	30s	-	11m	11m 40s
15	10 mins 31 secs	1m 58s	21s	12m 50s	16m 22s
16	10 mins 23 secs	12m 50s	1m 50s	25m 3s	12m 53s
17	10 mins 25 secs	1m 48s	1m 50s	14m 3s	16m 11s

Table 7.23: Mining Runtime of all experiments using Gemini 1.5-pro and default prompt

7.4.1 Experimental Approach

The following approach is defined to test the effect of the default prompt on Gemini 1.5-Flash performance in this work.

- **Initial Queue Task:** Send the KB relations to the LLM and apply the LLM evaluation afterward, corresponding to experiments 5 and 6 (no mining, though)
- **Evaluation Task:** Send the baseline discarded rules to the LLM, corresponding to experiment 2
- **Refinement Task:** Send the baseline rules to the LLM and apply LLM evaluation afterward, corresponding to experiments 3 and 4

7.4.2 Results

The results of the experimental approach above are described in this section. Section 8.1 details the rule quality for these experiments.

Initial Queue Task

Three requests were made using the default prompt, including the KB relations list, to test the improvement in the initial queue task. The results show that Gemini 1.5-Flash generates between 20-30 rules, significantly reducing the number of generated rules compared to the basic prompt.

When sending these rules to Gemini 1.5-Flash for contextual evaluation, the results show that most are evaluated to *true*. This means that if AMIE discards all the rules generated by Gemini 1.5-Flash using the default prompt in the initial queue task, which is considered the worst-case scenario, the LLM evaluation would evaluate most of these rules to *true*. Hence, they would be included in the mining result. This is similar to the result of experiment 6F.

Evaluation Task

Using the basic prompt, the results of experiment 2F show that the model evaluates most of the rules to *true*. Human inspection showed that the allowed rules have invalid context, which could indicate that Gemini 1.5-Flash does not fully understand the task, as discussed in Section 8.1. The default prompt discussed in Section 5.1 aims to clarify this task, leading to a better result using Gemini 1.5-Pro. To test the default prompt with Gemini 1.5-Flash, AMIE's list of discarded rules (1301) is sent to the model for contextual evaluation, corresponding to experiment 2 shown in Section 7.3.2. The results show that 850 rules are evaluated to *true* by the Gemini 1.5-Flash, meaning that 65% of the 1301 rules are outputted in the mining result. This is a better and more realistic result than having 94% of the rules evaluated to *true*, as when using the basic prompt.

Refinement Task

The improvement of the default prompt on the LLM refinement task can be measured by sending the baseline mining output (124 rules) to Gemini 1.5-Flash to discover new rules, potentially with relations from outside the KB. The results show that the model discovers 98 rules, compared to 120 rules discovered using the basic prompt in Appendix 10. Using the LLM evaluation, 91 of these rules are evaluated to *true*, which means that they would be included in the mining output, analog to the initial queue task rules explained above.

7.4.3 Outcome

The experimental approach designed to test the effectiveness of the setup of Gemini 1.5-Flash with the default prompt ensures that the same inputs are used to simulate experiments 2F, 3F, 5F, and 6F. The results of this subset of experiments allowed for expecting that the results of the other experiments using this setup would not improve. Specifically, the results presented above show that the improvements over Gemini 1.5-Flash with the basic prompt are minimal. Therefore, further experiments from Section

7.2 are not conducted using Gemini 1.5-Flash and the default prompts because no added value is expected. However, the discussion of the results of this setup, with a focus on the rule quality, is described in Section 8.1.

CHAPTER 8

Results Discussion

This chapter discusses in detail the results from Chapter 7, focusing on the objectives described in Section 5.5.1. This includes discussing the effectiveness of the LLM integration, the impact of each LLM task on the mining result, and the other tasks.

8.1 Rule Quality

The results in Section 7.3 focus on quantitatively showing the dependencies between LLM tasks, the effect of the request repetition concept on the mining result, and the effect of the model and prompt. In general, the results show that most of the rules generated by the LLM in the initial queue task and refinement task are not outputted without enabling the LLM evaluation. This means AMIE evaluation does not allow these rules in the output set, having low-quality metrics and low PCA confidence that do not pass the thresholds. This behavior could be due to one of two reasons:

- the rules are contextually correct, but there is not enough evidence for them in the KB.
- the rules are contextually incorrect, which, by extension, would imply that they do not have enough evidence, if any.

To address this, manual inspection is made on samples of the rules to check their contextual integrity. This section describes the rule quality of rules generated by both Gemini models used in this work, in the LLM initial queue task and refinement task. Section 8.2.3 describes the quality of rules evaluated by the LLM.

8.1.1 Gemini 1.5-Flash (basic prompt)

This model is used when running experiments 1-17 using the basic prompts (defined in Appendix 10). Manual inspection shows that most of the generated rules in both the LLM initial queue task and the refinement task are contextually incorrect. This impedes AMIE from outputting them, as there is insufficient evidence for most of these rules in the KB. This means that the rules generated by this model with the basic prompt in both tasks have no logical connection between the atoms. However, some of these rules are still outputted by AMIE, as shown in the following two examples. The former is the result of experiment 5, which contains one rule generated in the LLM initial queue task, and the latter is the result of experiment 7, which includes three generated rules by the refinement task:

$$actedIn(X, Y) \wedge hasGender(Y, Z) \Rightarrow hasGender(X, Z)$$

$$isMarriedTo(A, B) \wedge hasChild(B, C) \Rightarrow hasChild(A, C)$$
$$actedIn(A, B) \wedge hasGender(B, C) \Rightarrow hasGender(A, C)$$
$$hasChild(A, B) \wedge hasGender(B, C) \Rightarrow hasGender(A, C)$$

The rules in the above examples are contextually incorrect but still outputted by AMIE because enough evidence was found in the KB for them to pass the pruning strategy. This is a good example of how AMIE can still mine rules without logical implications.

In addition, the manual inspection of the results of other experiments identifies a pattern in generating the rules in both LLM tasks. This can also be seen in the above examples. It seems that Gemini 1.5-Flash uses two atoms to build a rule and uses a transitive relationship to simulate the logical connection between the atoms through the variables, i.e.,

$$A \rightarrow B \wedge B \rightarrow C \Rightarrow A \rightarrow C$$

Looking at the other rules generated but not outputted, most generated rules imply this pattern and include only two contextually irrelevant atoms, and most of them have invalid logical connections. This is why such rules are not outputted without enabling the LLM evaluation. Section 8.2 discusses how the LLM evaluation affects the rules generated by the Gemini models. The manual inspection of 30 rules shows that only two rules generated by this model contain a valid context, and the pattern from above is seen:

$$wasBornIn(A, B) \wedge hasOfficialLanguage(B, C) \Rightarrow hasOfficialLanguage(A, C)$$
$$hasWebsite(A, B) \wedge hasLanguageCode(B, C) \Rightarrow hasLanguageCode(A, C)$$

8.1.2 Gemini 1.5-Pro (default prompt)

In contrast to Gemini 1.5-Flash, the manual inspection of the rules generated by this model shows a notable improvement, where the rules outputted imply a correct context. Most of these rules, however, are not outputted by AMIE, as they usually do not have enough evidence in the KB. Therefore, enabling the LLM evaluation task to output these rules is crucial. An example of a rule generated by this model and outputted by AMIE:

$$exports(C, B) \wedge imports(A, B) \Rightarrow hasNeighbor(A, C)$$

According to Google, this model is optimized for various reasoning tasks and is most suitable for complex reasoning [Goo25]. This can be seen in the generated rules, where about 80% of the 30 rules that were manually inspected have a correct context.

8.1.3 Gemini 1.5-Flash (default prompt)

The default prompt introduces more detail for the LLM to use when generating responses. The results of the manual requests made to Gemini 1.5-Flash using the default prompt show that the quality of the rules generated by the initial queue task improved, having the relations used to build the rules relevant to each other, i.e., do not seem randomly picked from the list of KB relations. A pattern can still be identified in the logical connections between rule atoms.

The same does not apply to rules generated in the refinement task. Most of these rules are generated after a pattern and built using two contextually irrelevant relations. Therefore, a conclusion can be made that the default prompt improves the rule quality slightly.

8.2 LLM Effect

This section discusses the results of experiments 1-12, related to all objectives, and for both Gemini models used in this work. The results comparison is based on the baseline defined in Section 7.1.2.

8.2.1 LLM Initial Queue Task Discussion

Looking at the results of experiments 5, 7, 9, and 11, where the LLM initial queue task is enabled with no LLM evaluation, we can observe that most of the rules generated by both Gemini models by combining KB relations are not outputted.

In particular, Gemini 1.5-Flash, using the basic prompt, generates 359 rules, with an average of 59 rules generated per experiment, i.e., per request. Only a total of six rules are outputted, which corresponds to an average of 1-2 rules outputted per experiment. This means that only 1,67% of rules generated by the initial queue task using Gemini 1.5-Flash are outputted without enabling LLM evaluation, implying that the AMIE algorithm discarded most of them. This is due to the poor quality of the rules generated

by this model with the basic prompt, which is generated after a pattern and has an incorrect context, as explained in Section 8.1.

As for the Gemini 1.5-Pro, the rules generated by this task are much fewer, ranging between 10-15 rules per request, from which only 1-2 rules are outputted per experiment. This is the same percentage of rules outputted by the corresponding experiments with setup F, even though the generated rules have a correct context and higher quality, as explained in Section 8.1. This happens because the rules generated do not have enough evidence to be outputted by AMIE.

An important observation in this task is that AMIE does not apply its refinement operators (see Table 4.1) to build new rules. This is because these rules reached the maximum rule length threshold, an essential criterion for refining rules.

From the above observation and discussion, it is concluded that AMIE neither benefits from the rules generated by the LLM initial queue task for refinement nor from having a reasonable percentage of them in the mining result. This means that the LLM initial queue task does not bring any benefit when enabled alone.

Request Repetition Impact

The results from Tables 7.20 and 7.21 show that the Gemini 1.5-Flash generates 500+ rules when the LLM initial queue task request is repeated 8 times, from which a subset of rules is selected depending on the selection criteria. The most interesting questions here are whether the duplicated rules improve the performance of the initial queue task and whether the vast amount of generated rules could include higher-quality rules outputted by AMIE. The results and answers to the former question depend on the used model.

For Gemini 1.5-Flash, the results show that the duplicated rules are not outputted without LLM evaluation, which evaluated most of them to *true* once enabled. Using Gemini 1.5-Pro, a small number of duplicated rules are outputted by AMIE, i.e., 11% in experiment 13. However, half of the duplicated rules are outputted by LLM evaluation, with good rule quality as explained in Section 8.1. This means that the duplicated rules improve the performance of the initial queue task.

As for the other selection criteria, the number of rules outputted proportionally grows with the number of generated rules. This applies to both models. Therefore, to answer the second question, the number of generated rules improves the number of LLM rules in the mining result and includes higher-quality rules in the output.

8.2.2 LLM Refinement Task Discussion

This task uses the rules mined by AMIE to generate new rules, possibly using atoms from outside the KB. The results of experiment 3, where this task uses the 124 rules outputted by AMIE to generate rules, show that on average 5% of the rules generated by Gemini 1.5-Flash with the basic prompt are outputted without LLM evaluation, compared to 3% of the rules generated by Gemini 1.5-Pro.

Experiments 7 and 11 results show that the number of rules generated by the refinement task depends highly on the number of input rules, i.e., the number of rules mined in the first iteration. This means that if only the rules generated by the LLM in the initial queue task are in the initial queue (experiment 7), the mining result will not be promising, as the refinement task tries to discover new rules based on a small number of rules inputted. In conclusion, the results of experiment 11 are much better, as also the 124 rules outputted by AMIE are sent for refinement, where 20 times more rules were generated by Gemini 1.5-Flash, and correspondingly, 9 times more rules with Gemini 1.5-Pro.

Despite the high number of rules generated by the LLM refinement task, the number of rules outputted remains small without LLM evaluation. An explicit dependency can be seen between the initial queue task and the refinement task, where the former's effectiveness greatly affects the latter's performance.

Rules with New Atoms

To fulfill objective 5 (see Section 5.5.1), the rules generated by the LLM refinement task are examined. To conclude whether the refinement task effectively finds rules with atoms from outside the KB, experiment 4, in which the 124 baseline rules are sent to the LLM, is manually repeated, ensuring a statistical analysis of the results. The results show that Gemini 1.5-Pro finds, on average, 15 new atoms used in the generated rules. Comparatively, Gemini 1.5-Flash finds, on average, 110 new atoms. This is because Gemini 1.5-Flash generates much more rules than Gemini 1.5-Pro.

As expected, the results show that AMIE discards all the rules with new atoms, as it cannot find enough evidence for them in the KB. Therefore, LLM evaluation is the only way to have rules with new atoms in the mining result.

8.2.3 LLM Evaluation Task Discussion

The results of the experiments where LLM evaluation is enabled are discussed in this section. First, the performance of the LLM when evaluating the context of rules is discussed, including whether the rules are correctly evaluated, i.e., *true*-evaluated rules have valid context.

Evaluation Performance

Experiment 2 shows that setup F and P evaluate 94% and 26% of the rules discarded by AMIE to *true*, respectively. Additionally, Gemini 1.5-Flash with the default prompt evaluates 65% of these rules to *true*. The Manual inspection shows that rules evaluated by Gemini 1.5-Flash with both prompts are mostly contextually incorrect, as the model seems to guess if a rule has a correct context. This could be traced back to the optimization of this model. According to Google, this model tries to generate as much content as possible in the shortest time possible [Goo25], an issue that impedes accurate assessment of each rule. The runtime results from Table 7.22 show that Gemini 1.5-Flash takes 3 minutes to

evaluate the 1301 rules discarded by AMIE in experiment 2 (baseline discarded rules), as shown in the following rule which is evaluated to *true* using this model with the basic prompt:

$$hasWikipediaCategory(A, F) \wedge hasWikipediaCategory(B, F) \Rightarrow hasWonPrize(A, B)$$

Gemini 1.5-Pro, however, takes a longer time to evaluate rules. Table 7.23 shows that this model takes about 21 minutes to evaluate the same 1301 rules, which is 7 times more than Gemini 1.5-Flash. The evaluated rules, however, are much less, as it evaluates 26% rules to *true*, which are found in the mining result. Manual inspection of the rule quality shows that most of them contain a valid context, which makes this model suitable for this LLM task. This means that Gemini 1.5-Pro correctly identifies the logical connections between the atoms and the relevancy of the relations used in the rule's head and body. Some rules, however, are still evaluated by Gemini 1.5-Pro to *true* although they contain a vague context, as shown in the following example:

$$hasWikipediaLinkTo(E, A) \wedge hasWikipediaLinkTo(E, B) \Rightarrow dealsWith(A, B)$$

The reason why this rule was evaluated to *true* by Gemini 1.5-Pro is assumed to be due to the word "*Link*" contained in the relation "hasInternalWikipediaLinkTo", which implies that if two entities are linked through a Wikipedia link, then they both 'deal' with each other.

An interesting observation of the LLM evaluation tasks is that it fails to correctly evaluate some rules generated by other tasks. The Gemini 1.5-Pro model with the default prompt generated the following rules, which are later evaluated for contextual correctness. The result was that only one rule (highlighted below) was outputted.

$$livesIn(A, B) \wedge isLocatedIn(C, B) \Rightarrow isCitizenOf(A, C)$$
$$worksAt(A, B) \wedge owns(C, B) \Rightarrow dealsWith(A, C)$$
$$graduatedFrom(A, B) \wedge hasCapital(C, B) \Rightarrow livesIn(A, C)$$
$$isLeaderOf(A, B) \wedge isCitizenOf(C, B) \Rightarrow isPoliticianOf(C, A)$$
$$hasChild(A, B) \wedge isMarriedTo(C, A) \Rightarrow hasFamilyName(B, C)$$
$$\mathbf{directed}(A, B) \wedge actedIn(C, B) \Rightarrow isKnownFor(A, C)$$
$$wroteMusicFor(A, B) \wedge produced(C, B) \Rightarrow worksAt(A, C)$$
$$hasOfficialLanguage(B, C) \wedge livesIn(A, B) \Rightarrow isInterestedIn(A, C)$$
$$wasBornIn(A, B) \wedge hasCapital(C, B) \Rightarrow isCitizenOf(A, C)$$
$$diedIn(A, B) \wedge isLocatedIn(C, B) \Rightarrow wasBornIn(A, C)$$

LLM Evaluation Impact

The discussion from Section 8.2.1 concludes that the rules generated by the LLM initial queue task are of no benefit without enabling the LLM evaluation, which is reflected in the results of experiments 6 and 10 in Tables 7.6 and 7.7. It is shown that 99% of the rules generated by the LLM initial queue task are evaluated to *true* by Gemini 1.5-Flash, hence found in the mining result. The same does not apply to Gemini 1.5-Pro, which evaluates the rules more carefully, leading to less than 40% of the rules generated by the initial queue task evaluated to *true*. Therefore, the performance of the LLM evaluation task directly affects the initial queue task.

Similarly, most rules generated by the refinement task in experiment 4 are outputted due to the LLM evaluation. Specifically, Gemini 1.5-Flash evaluates 38% of the rules generated by the refinement task to *true*. Gemini 1.5-Pro evaluates 42% of the corresponding rules to *true*.

Conclusively, the LLM evaluation task is crucial for outputting rules generated by the LLM, where the quality of the evaluated rules depends primarily on the model used.

8.3 Runtime and AMIE Integration

Integrating the three LLM tasks in AMIE is implemented to separate the LLM effect from AMIE; if no LLM task is enabled, AMIE runs without any influence from the LLM, which is the baseline AMIE. When enabled, the LLM integration affects mainly the runtime of the algorithm, as shown in Tables 7.22 and 7.23 for both Gemini models used in this work.

8.3.1 LLM Tasks Runtime

The results of experiments 1-12 show that the time spent in the LLM initial queue task is minimal, with a maximum of half a minute, independent from the model used. Similarly, the LLM refinement time runtime does not exceed 2 minutes with Gemini 1.5-Pro because of the relatively small number of rules this model outputs using the LLM evaluation in the first iteration. In contrast, Gemini 1.5-Flash outputs many more rules using the LLM evaluation compared to Gemini 1.5-Pro, leading to a higher runtime of the refinement task using this model. The highest time spent by the LLM is on the evaluation task, which is up to 6 minutes for Gemini 1.5-Flash and 25 minutes for Gemini 1.5-Pro. Although Gemini 1.5-Flash is faster than Gemini 1.5-Pro [Goo25], this disparity in the runtime of the evaluation task using both models remains high, and it can be traced back to other factors, such as the number of rules to be evaluated, and more importantly, the implementation of the Gemini client, which batches rules and throttles requests as explained in Section 6.1.3. The Gemini client also affects the LLM refinement task, where rules are also batched, and the recurrent requests are throttled.

Gemini 1.5-Pro uses a smaller batching number, i.e., 20 rules per batch, which leads to more batches and, hence, more requests to Gemini. Additionally, the number of free

requests allowed by this model per minute is 2, which means that every third request will be waiting until the minute since the first request passes. This explains the high runtime of the evaluation task. Gemini 1.5-Flash, however, sends 150 rules per batch, and the number of free requests allowed is 15, in addition to its fast response generation, which is the reason behind its runtime.

The results of experiments 13-17 in Tables 7.22 and 7.23 show the runtime of each LLM task when using the request repetition concept. We can see that the time spent in the initial queue task is higher than in the other experiments, which is traced back to the batching and request throttling, as explained above.

8.3.2 Side Effects and Improvements

The discussion above shows that the runtime of the LLM tasks depends on the Gemini client. The restrictions of a Gemini model are the main reasons why Gemini clients batch rules and throttle requests, as the former is due to the maximum output tokens restriction, and the latter is due to the maximum requests per minute restriction. To avoid the effect of the Gemini client, the batch number and maximum requests per minute threshold used for throttling requests can be raised, leading to a better runtime of each LLM task, especially the evaluation task. This is related to the cost of using Gemini, which charges each request outside the restrictions defined by each model.

CHAPTER 9

Conclusion and Future Work

This thesis investigates the ability of the LLM to support rule mining. Specifically, the integration of LLM into AMIE, a rule miner that mines horn rules from knowledge bases, is designed and implemented by exploring different phases in which the LLM can assist. Three phases of AMIE are qualified for LLM assistance. The first phase is the initialization phase, where AMIE builds a queue of simple implication rules from all relations in the knowledge base. Here, the LLM can use the knowledge base relations to generate candidate rules to complement the queue. The second phase is the evaluation phase, where AMIE decides if a rule can be outputted based on the PCA confidence measurement of this rule on the knowledge base. The context evaluation score of the rule is introduced as an additional measurement to output a rule, which the LLM determines. Finally, an additional rule refinement procedure is implemented within the refinement phase, where the LLM discovers new rules based on the context of the already mined rules. Several techniques have been implemented to ensure the highest possible effectiveness of the LLM in rule mining while keeping a balance between the LLM costs and computational overhead. This includes repeating the request N times to enable statistically acceptable results, prompting with different models, and optimization techniques such as batching and throttling requests to the LLM.

To objectively test the integration of the LLM within AMIE, comprehensive experiments were conducted on the YAGO2 knowledge base to ensure the effect of each LLM task on the mining is tested individually, as well as its effect on the other LLM tasks. The experiments also include testing this work's various techniques and optimizations using different LLMs and prompts. The results show that the LLM Gemini 1.5-Flash using the basic prompt is unsuitable for qualitatively assisting the rule mining process because of its fine-tuning and optimization to prefer fast response generation over reasoning, leading to generating many rules that don't have valid context, which contain a pattern and inaccurate evaluation of rules. When using the default prompt, fewer rules are generated by this LLM, and they have better quality than the ones generated using the basic

prompt. Still, the pattern in rules generation can be identified, and the evaluation of rules remains mostly inaccurate. The LLM Gemini 1.5-Pro and the default prompt result in higher-quality generated rules, and the rule evaluation is more accurate. However, this model generates fewer rules, leading to a slight quantitive improvement in AMIE, yet highly qualitative.

In summary, the Gemini 1.5-Pro model is suitable for assisting AMIE in rule mining, where the costs of using it can be mitigated using batching and throttling request techniques. This, however, introduces a higher runtime of the algorithm, which implies a trade-off between costs and computational overhead.

9.1 Future Work

The research presented in this thesis is the foundation of implementing LLM-assisted rule miners that complement other rule miners. Future work can include designing other approaches to generate rules, e.g., iteratively generating rules focusing on a particular KB relation rather than sending all KB relations simultaneously. Additionally, future work can unleash the full power of the LLM without the techniques that introduce computational overhead.

CHAPTER 10

Appendix

10.1 Basic Prompt

This lists the basic prompts used in Setup F, as described in Section 7.2.3.

10.1.1 Initial Queue Task Old Prompt

Prompt

In the context of Rule Learning, build closed candidate rules from the knowledge base schema below:
// list of relations
Short answer only. Use only the atoms provided above and variables following the regex `^\?[a-z][0-9]$`
Example on the format of a rule:
?a p0:relationX ?b ?c p0:relationY ?a => ?c p0:relationZ ?a
where ?a p0:relationX ?b is equivalent to Datalog atom p0:relationX(?a, ?b).

10.1.2 Evaluation Task Basic Prompt

Prompt

In the context of Rule Mining, evaluate the current rules for context:

// list of rules to be evaluated
Use boolean true or false for the field "context" in the json format.

10.1.3 Refinement Task Basic Prompt

Prompt

In the context of Rule Learning, use the following list of relations to generate new rules by incorporating domain knowledge. You can create new relations that work in the domain.
// list of rules to be refined
Short answer only. Use variables following the regex `^\?[a-z][0-9]$`
Example on the format of a rule:
?a p0:relationX ?b ?c p0:relationY ?a => ?c p0:relationZ ?a
where ?a p0:relationX ?b is equivalent to Datalog atom p0:relationX(?a, ?b).

10.2 Algorithms

This lists additional algorithms used in this work.

10.2.1 Throttling Requests Algorithm

Algorithm 10.1: Request Throttling Algorithm

Input: MaxRequests M, Time Window T (in milliseconds)
Data: Deque *requestTimestamps*
Function *ThrottleRequests()*:
  now ← `System.currentTimeMillis` ();
  while *requestTimestamps is not empty* ***and*** *now* − `peekFirst`(*requestTimestamps*) > T **do**
    `pollFirst` (requestTimestamps);
  end
  if |*requestTimestamps*| ≥ M **then**
    earliestTimestamp ← `peekFirst`(*requestTimestamps*);
    waitTime ← T − (*now* − *earliestTimestamp*);
    Thread.sleep(waitTime);
  end
Function *AddToTimestamp()*:
  `addLast` (requestTimestamps, `System.currentTimeMillis` ());

Übersicht verwendeter Hilfsmittel

Generative AI tools, specifically ChatGPT, were used to generate Latex code for tables based on the data in Excel files. Additionally, ChatGPT was used to translate logical rules from AMIE formatting into the formatting presented in the thesis.

Grammarly AI was also used to maintain grammar and paraphrasing.

List of Figures

List of Tables

List of Algorithms

Bibliography

[Ant25] Anthropic. Claude ai, 2025. Accessed: 2025-03-02.

[Bar77] Jon Barwise. An introduction to first-order logic. In Jon Barwise, editor, *HANDBOOK OF MATHEMATICAL LOGIC*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, pages 5–46. Elsevier, 1977.

[BCM04] Gene Bellinger, Durval Castro, and Anthony Mills. Data, information, knowledge, and wisdom, 2004.

[BP25] David Beckett and Eric Prud'hommeaux. Rdf 1.2 turtle - terse rdf triple language. https://www.w3.org/TR/rdf12-turtle/, 2025. Accessed: 2025-01-03.

[CGWJ16] Yang Chen, Sean Goldberg, Daisy Zhe Wang, and Soumitra Siddharth Johri. Ontological pathfinding. In *Proceedings of the 2016 international conference on management of data*, pages 835–846, 2016.

[Cha24] Eben Charles. Data validation techniques for ensuring data quality. *Research Gate*, 09 2024.

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. `https://arxiv.org/abs/1810.04805`, 2019.

[DIG25] DIG Team. AMIE Github Repository, 2025. Accessed: 2025-02-23.

[DMM00] S. Decker, P. Mitra, and S. Melnik. Framework for the semantic web: an rdf tutorial. *IEEE Internet Computing*, 4(6):68–73, 2000.

[Gen05] Michael Genesereth. The digital decision box. `http://ggp.stanford.edu/notes/ddb.html`, 2005. Accessed: 2025-01-28.

[gen23] The key differences between intelligence and knowledge: A comparative analysis. https://genialpha.com/the-key-differences-between-intelligence-and-knowledge-a-comparative-analysis/, 2023. Accessed: 2025-01-02.

[Goo25] Google AI. Gemini API Documentation, 2025. Accessed: 2025-02-23.

[GRAS17] Luis Galárraga, Simon Razniewski, Antoine Amarilli, and Fabian M. Suchanek. Predicting completeness in knowledge bases. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, page 375–383, New York, NY, USA, 2017. Association for Computing Machinery.

[GTHS13] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, page 413–422, New York, NY, USA, 2013. Association for Computing Machinery.

[GTHS15] Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. Fast rule mining in ontological knowledge bases with amie+. *The VLDB Journal*, 24(6):707–730, December 2015.

[GW98] Robert Gaizauskas and Yorick Wilks. Information extraction: beyond document retrieval. *Journal of Documentation*, 54(1):70–105, 1998.

[HMR21] Oliver Haas, Andreas Maier, and Eva Rothgang. Rule-based models for risk estimation and analysis of in-hospital mortality in emergency and critical care. *Frontiers in Medicine*, 8:785711, 2021.

[HSBW13] Johannes Hoffart, Fabian M Suchanek, Klaus Berberich, and Gerhard Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial intelligence*, 194:28–61, 2013.

[HYM+25] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Trans. Inf. Syst.*, 43(2), January 2025.

[LGS20] Jonathan Lajus, Luis Galárraga, and Fabian Suchanek. Fast and exact rule mining with amie 3. In *The Semantic Web: 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31–June 4, 2020, Proceedings*, page 36–52, Berlin, Heidelberg, 2020. Springer-Verlag.

[LJX+23] Linhao Luo, Jiaxin Ju, Bo Xiong, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. Chatrule: Mining logical rules with large language models for knowledge graph reasoning. *arXiv preprint arXiv:2309.01538*, 2023.

[Mah20] Batta Mahesh. Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, 9(1):381–386, 2020.

[MR94] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 1994.

[Mun25] Emir Munoz. Relation cardinality in knowledge bases. https://emunoz.org/kb-cardinality, 2025. Accessed: 2025-01-31.

[OMP18] Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. Robust discovery of positive and negative rules in knowledge bases. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1168–1179, 2018.

[Ope25] OpenAI. Chatgpt overview, 2025. Accessed: 2025-03-02.

[Pok15] Jaroslav Pokornỳ. Graph databases: their power and limitations. In *Computer Information Systems and Industrial Management: 14th IFIP TC 8 International Conference, CISIM 2015, Warsaw, Poland, September 24-26, 2015, Proceedings 14*, pages 58–69. Springer, 2015.

[Rad18] Alec Radford. Improving language understanding by generative pre-training. *IEEE Journal of Selected Topics in Signal Processing*, 2018.

[SDEW10] Stefan Schoenmackers, Jesse Davis, Oren Etzioni, and Daniel S Weld. Learning first-order horn clauses from web text. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 1088–1098, 2010.

[SSPA+12] Fatimah Sidi, Payam Hassany Shariat Panahy, Lilly Suriani Affendey, Marzanah A. Jabar, Hamidah Ibrahim, and Aida Mustapha. Data quality: A survey of data quality dimensions. In *2012 International Conference on Information Retrieval & Knowledge Management*, pages 300–304, 2012.

[TBL+19] Yao-Hung Hubert Tsai, Shaojie Bai, Paul Pu Liang, J Zico Kolter, Louis-Philippe Morency, and Ruslan Salakhutdinov. Multimodal transformer for unaligned multimodal language sequences. In *Proceedings of the conference. Association for computational linguistics. Meeting*, volume 2019, page 6558. NIH Public Access, 2019.

[Vas17] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[Wis25] Wisecube AI. Knowledge Graphs vs. Relational Databases: Everything You Need to Know. https://www.wisecube.ai/blog/knowledge-graphs-vs-relational-databases-everything-you-need-to-know/, 2025. Accessed: 2025-01-03.

[WSS+22] Lianlong Wu, Emanuel Sallinger, Evgeny Sherkhonov, Sahar Vahdati, and Georg Gottlob. Rule learning over knowledge graphs with genetic logic programming. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 3373–3385. IEEE, 2022.

[YYC+22] Gene-Ping Yang, Sung-Lin Yeh, Yu-An Chung, James Glass, and Hao Tang. Autoregressive predictive coding: A comprehensive study. *IEEE Journal of Selected Topics in Signal Processing*, 16(6):1380–1390, 2022.

[ZB03] Qiankun Zhao and Sourav S. Bhowmick. Association rule mining: A survey. *Technical Report, CAIS, Nanyang Technological University, Singapore*, 2003.