CentraleSupélec

université PARIS-SACLAY

Big Data Research Project

Neurosymbolic Reasoning with RNNLogic

Olha Baliasina
Nima Kamali Lassem
Adrian Patricio

Github Link [1]

Supervisor: Laura Isabella Forero Camacho - CentraleSupélec

[1] Access our Github here: https://github.com/NimakamaliLassem/BDRP

Contents

Chapter 1

Introduction

Knowledge graphs (KGs) store structured facts about a domain as triples of the form (h, r, t), where h and t are entities and r is a relation. This representation is widely used whenever a system needs to combine heterogeneous information and support relational queries, for example in search, recommendation, and question answering pipelines [10]. In practice, however, real KGs are almost always incomplete: they are built from multiple sources, updated over time, and often rely on automatic extraction, so many true facts are missing.

A standard way to address this incompleteness is Knowledge Graph Completion (KGC), also referred to as link prediction. Given a query such as $(h, r, ?)$ or $(?, r, t)$, the goal is to rank candidate entities and recover missing triples. The central challenge is that KGC models must be both accurate and usable: in many applications, it is not enough to produce a prediction, we also want to understand why the model believes that the prediction is plausible.

1.1 The Interpretability-Accuracy Trade-off

The field of KGC has historically been defined by a trade-off between predictive performance and model interpretability. This trade-off manifests through two dominant paradigms:

1. **Embedding-based Methods:** Approaches such as TransE, RotatE, and ComplEx achieve state-of-the-art predictive accuracy by learning low-dimensional vector representations for entities and relations. While effective, these models operate as "black boxes," performing abstract mathematical operations in high-dimensional space that yield accurate predictions without offering human-readable explanations.

2. **Symbolic Rule-based Methods:** Conversely, symbolic approaches aim to discover explicit first-order logic rules directly from the graph structure, such as $\forall X, Y, Z : \text{parent}(X, Y) \wedge \text{parent}(Y, Z) \rightarrow \text{grandparent}(X, Z)$. Although these rules are perfectly interpretable and generalize well to unseen data, symbolic methods face significant computational challenges due to the exponentially large search space of possible rules and often suffer from lower accuracy compared to embedding models.

This project addresses this dichotomy by investigating a central research question: *How can we perform accurate prediction on knowledge graphs while effectively providing clear, human-readable explanations?*

1.2 Project Objectives

To answer this question, we leverage RNNLogic [17] as our primary foundation. RNNLogic represents a significant advance in neurosymbolic reasoning that bridges the gap between neural and symbolic approaches by treating logic rules as latent variables. However, we identify specific limitations in the original framework, particularly regarding the sequential bottleneck of its LSTM-based generator and the indirect nature of its Expectation-Maximization (EM) optimization.

1.2.1 General Goals

The overarching goal of this project is to enhance the scalability, expressiveness, and convergence speed of the RNNLogic framework. We aim to address specific architectural and optimization limitations of the original framework to enable:

- **Efficient Short-Range Rule Learning:** Optimizing the model's ability to learn concise logical rules on smaller knowledge graph datasets through more parameter-efficient architectures.
- **Improved Optimization:** Enhancing the Expectation-Maximization (EM) training loop with reinforcement learning-inspired mechanisms to achieve more stable and efficient rule learning.
- **Enhanced Interpretability:** Incorporating mechanisms that provide transparency into the reasoning process, enabling better understanding of how learned rules contribute to predictions and supporting validation of logical patterns.

1.2.2 Specific Goals

To achieve these general improvements, we define three specific technical objectives:

1. **Architectural Refinement with GRU:** We replaced the standard LSTM-based rule generator with Gated Recurrent Units (GRU). Our hypothesis is that given the short nature of logical rules (default 3-hop chains) and the relatively small scale of benchmark datasets (UMLS, Countries-S3), GRU's simplified gating mechanism would provide comparable or improved performance while reducing computational overhead and parameter count compared to LSTM.
2. **RL-Inspired EM Optimization:** We enhanced the original EM optimization framework with reinforcement learning-inspired techniques. Specifically, we developed a value-guided mechanism to improve rule quality assessment and selection during training, enabling the model to learn more effective logical rules by providing better guidance signals during the expectation and maximization steps.
3. **Attention-Based Interpretability:** We integrated attention mechanisms into the rule generator architecture to increase model interpretability. By analyzing attention weights, we can identify which particular relations in the learned rules contribute most significantly to final predictions, providing explicit explanations for the model's reasoning process and enabling validation of learned logical patterns.

1.3 Report Structure

The remainder of this report is structured as follows:

- **Chapter 2** reviews the related work in Knowledge Graph Completion, positioning RNNLogic within the landscape of embedding, rule-based, and neurosymbolic approaches.
- **Chapter 3** provides the technical background on the foundational RNNLogic framework, detailing its mining, generation, and reasoning phases.
- **Chapter 4** presents our novel methodology, detailing the mathematical foundations of the Mamba and GRU backbones, as well as the Value-Guided RL algorithm.
- **Chapter 5** discusses the experimental setup and presents a comparative evaluation of our models against baselines across standard benchmarks.
- **Chapter 6** concludes the report and outlines perspectives for future work.

Chapter 2

Related Work

Knowledge Graph Completion (KGC) is commonly framed as a link prediction task: given a query of the form $(h, r, ?)$ or $(?, r, t)$, a model ranks candidate entities to recover missing triples. Existing approaches can be broadly grouped by the representational trade-off they make between predictive accuracy, scalability, and interpretability.

In this chapter we review four major paradigms for KGC: embedding models, symbolic and statistical rule learners, reinforcement-learning path reasoners, and neurosymbolic methods. We pay particular attention to the neurosymbolic family, where we identify three distinct integration strategies and compare their characteristics. Through this analysis, we motivate why RNNLogic [17] occupies a particularly suitable position for our research objectives and establish the context for our contributions.

2.1 Embedding-based Link Prediction

Embedding methods represent entities and relations as vectors (or structured tensors) and learn a scoring function $f(h, r, t)$ that assigns high scores to plausible triples. These models tend to achieve strong predictive performance but provide limited transparency, since decisions are made through continuous computations rather than explicit rules.

2.1.1 Translation-based Models

Translation-based models treat each relation as a vector translation in the embedding space, enforcing that for a true triple (h, r, t) the translated head embedding lands close to the tail embedding.

TransE [2] is the foundational model of this family. It embeds entities and relations as vectors in $\mathbb{R}^d$ and scores plausibility via $\|h+r-t\|$, using L1 or L2 norms, so that the model minimizes this distance for valid triples. The resulting translation assumption $h + r \approx t$ naturally captures composition ($r_1+r_2 \approx r_3$) and antisymmetry ($h+r \neq t$ generally implies $t + r \neq h$), but it fails on 1-to-N, N-to-1, and symmetric relations. When multiple valid tails t_1, t_2 exist for the same head and relation, the model forces $t_1 \approx t_2$, collapsing distinct entities into the same region of the embedding space [2]. Subsequent extensions such as TransH, TransR, and TransD address this by introducing relation-specific projections or hyperplanes, at the cost of additional parameters.

2.1.2 Semantic Matching Models

Semantic matching models score triples by measuring compatibility between entity and relation embeddings through similarity or bilinear functions, rather than interpreting relations as geometric transformations.

DistMult [26] represents each relation r as a diagonal matrix $\mathrm{diag}(\mathbf{r})$ and scores a triple as

$$f(h, r, t) = \mathbf{h}^\top \mathrm{diag}(\mathbf{r})\, \mathbf{t} = \sum_{i=1}^{d} h_i\, r_i\, t_i.$$

This yields an efficient model with $O(d)$ parameters per relation that works well for symmetric patterns, but since $f(h, r, t) = f(t, r, h)$, it cannot model asymmetric relations such as *parent-of* vs. *child-of*.

ComplEx [24] addresses this limitation by extending DistMult into the complex vector space $\mathbb{C}^d$. The scoring function becomes

$$f(h, r, t) = \mathrm{Re}\big(\langle \mathbf{h}, \mathbf{r}, \overline{\mathbf{t}} \rangle\big) = \mathrm{Re}\left(\sum_{i=1}^{d} h_i\, r_i\, \overline{t_i} \right),$$

where $\overline{\mathbf{t}}$ denotes complex conjugation. The conjugation on $\mathbf{t}$ breaks the symmetry and enables modeling asymmetric and anti-symmetric relations while retaining the parameter efficiency of DistMult.

2.1.3 Rotation-based Models

Rotation-based models represent relations as rotations in the complex plane, so that for a valid triple (h, r, t) the relation rotates the head embedding to align with the tail.

RotatE [22] embeds entities as complex vectors $\mathbf{h}, \mathbf{t} \in \mathbb{C}^d$ and constrains each relation $\mathbf{r} \in \mathbb{C}^d$ to lie on the unit circle ($|r_i| = 1$ for all i). A valid triple satisfies $\mathbf{t} \approx \mathbf{h} \circ \mathbf{r}$, where $\circ$ is the element-wise (Hadamard) product, and the distance-based score is $d_r(h, t) = \|\mathbf{h} \circ \mathbf{r} - \mathbf{t}\|_p$. By parameterizing $r_i = e^{i\theta_{r,i}}$, RotatE can model symmetry, inversion, and composition through constraints on the rotation angles $\theta_{r,i}$.

RotatE is directly relevant to our work because RNNLogic uses it as the base embedding model for path scoring in the embedding-enhanced predictor [17]. We describe this integration in detail in the background chapter.

2.1.4 Limitations of Embedding Approaches

While embedding methods achieve strong predictive accuracy, they are fundamentally black-box predictors. The learned vectors capture logical patterns implicitly, but they do not expose human-readable rules or discrete reasoning chains that explain why a particular triple is predicted. They cannot express their knowledge as symbolic rules (e.g. Horn clauses), and it is generally difficult to trace a prediction back to a small set of understandable reasoning steps. Handling complex logical structures such as nested quantifiers or rule hierarchies also remains challenging within a purely continuous framework. These limitations motivate the exploration of rule-based approaches that learn explicit multi-hop rules, or hybrid neurosymbolic approaches that aim for both strong prediction and interpretable reasoning.

2.2 Rule Learning and Statistical Approaches

Rule-based and statistical methods operate directly on the symbolic structure of the graph. They aim to learn human-readable patterns such as Horn clauses, supporting explanations

of the form "(h, r, t) holds because rule ρ is satisfied by these supporting facts." Their main advantage is interpretability: each prediction can be justified by a specific, auditable chain of evidence. Their main difficulty is search, since the number of candidate rules grows rapidly with rule length and graph branching, and sparse graphs provide weak statistical evidence for individual rules.

2.2.1 Path Ranking Algorithm

The Path Ranking Algorithm (PRA) [11] treats reasoning as learning over relational paths. For a target relation r, PRA enumerates paths between entity pairs, treats each distinct path pattern as a binary feature, and trains a classifier over these features to predict whether (h, r, t) holds. This produces weighted rules of the form

$$r(h, t) \Leftarrow r_1(h, x_1) \wedge \cdots \wedge r_\ell(x_{\ell-1}, t),$$

which are intuitive and explainable. However, the number of distinct paths grows exponentially with length, making enumeration expensive. PRA also struggles on sparse graphs where few or no paths connect many entity pairs, and it does not easily generalize beyond simple chain-shaped rule templates.

2.2.2 Probabilistic Logic: Markov Logic Networks

Markov Logic Networks (MLNs) [18] combine first-order logic with Markov random fields. An MLN consists of a set of weighted first-order formulas (w_i, φ_i) for $i = 1, \ldots, m$, where each weight w_i reflects the strength of the corresponding constraint. Grounding these formulas over all entities yields a Markov network whose distribution favours worlds that satisfy the more heavily weighted formulas. In principle, this framework can express rich relational dependencies and uncertainty in a unified way. In practice, however, both grounding and inference are computationally expensive due to the potentially enormous number of ground atoms and clauses, and scaling MLNs to large knowledge graphs remains a significant challenge.

2.2.3 Neural Logic Programming

NeuralLP [27] represents rule application as sequences of differentiable matrix operations over adjacency matrices. Rule templates of a fixed maximum length are encoded, and their weights are learned end-to-end via gradient descent. This allows simultaneous learning of rule structures (through attention over relational operators) and their weights. DRUM [19] extends this idea with bidirectional recurrence over relations, enabling it to capture more complex multi-hop patterns.

These differentiable approaches reduce the need for explicit discrete search, but they can require repeated and costly large matrix multiplications over adjacency tensors, which becomes prohibitive on large knowledge graphs. Moreover, although learning is differentiable, the effective rule search space remains exponentially large in rule length, and extracting clean, human-readable rules from the learned parameters is non-trivial.

2.2.4 Limitations of Rule-based Approaches

Rule-based and statistical methods offer strong interpretability and can capture relational structures that embedding models miss, but they face persistent challenges. The candidate rule space grows exponentially with rule length, making exhaustive search infeasible for longer reasoning chains. Both inference and learning tend to be expensive on large knowledge graphs. Many approaches depend on observed paths, meaning that in sparse graphs useful rules may not appear frequently enough to be discovered or reliably weighted. Finally, jointly learning rule structures and their weights is difficult and often leads to unstable or slow training, particularly in neural or differentiable formulations. These limitations motivate hybrid approaches that retain the interpretability of rules while mitigating the search and optimization difficulties.

2.3 Reinforcement Learning Approaches

Reinforcement Learning (RL) methods formulate link prediction as a sequential decision-making process: an agent starts at a query entity and repeatedly selects outgoing edges to traverse, aiming to reach a correct answer entity. Policy-based methods learn a policy that directly selects the next action, while value-based methods learn a value function that estimates expected return and use it to guide decisions.

Many classic systems in this space rely on policy gradients. MINERVA [8] trains pathfinding agents with REINFORCE to search for reasoning paths connecting query entities to correct answers. MultiHopKG [13] introduces reward shaping with knowledge graph embeddings to mitigate the sparse reward problem. M-Walk [20] employs Monte Carlo Tree Search for more structured exploration.

A common difficulty across these methods is reward sparsity: meaningful feedback often only arrives when a path successfully reaches an answer entity. This has led to various forms of reward shaping and guided search. An important insight for our project is that RL can be applied not just to traverse the graph, but also to generate or select rules under a reward derived from downstream reasoning quality. This perspective directly connects to our replacement of the standard EM training loop in RNNLogic with a value-guided optimization procedure.

2.4 Neurosymbolic Approaches

Neurosymbolic approaches combine the learning capability and noise tolerance of neural networks with the reasoning capability and interpretability of symbolic logic, positioning themselves as a middle ground between embedding-only models and purely symbolic rule learners. Over the past several years, a range of such methods has been proposed for knowledge graph reasoning. To clarify the landscape and understand where RNNLogic fits, we organize them into three broad categories according to how they integrate the neural and symbolic components: compositional rule learning methods, LLM-augmented rule mining, and latent-variable rule generation.

2.4.1 Compositional Rule Learning

One natural strategy for managing the combinatorial explosion of candidate rules is to impose structural constraints on how rules are built. Instead of treating each rule as a flat sequence of relations, compositional methods decompose rules into smaller, reusable components and assemble them hierarchically.

Neural Compositional Rule Learning (NCRL) [3] represents logic rules as hierarchical compositions rather than flat chains. Rule bodies are decomposed into small sub-rules that can be recombined across different rules. This sharing of atomic components enables systematic generalization to unseen relational patterns and reduces redundancy, allowing NCRL to scale to large knowledge graphs while maintaining good generalization. The core idea is that many useful rules share common sub-patterns (for example, the sub-rule "works-at $\wedge$ located-in" might appear in multiple longer chains), and that explicitly representing this shared structure improves both efficiency and coverage.

RLogic [4] takes a related but distinct approach by learning rules at the schema level through representation learning. Instead of training a single large sequential model, RLogic breaks the rule model into small atomic components that are combined recursively. This recursive design pushes deductive reasoning deeper into the learning process, avoids the bottleneck of a single sequential controller, and helps learn rules that generalize across different relations and schemas.

Both NCRL and RLogic produce interpretable rules and handle the combinatorial challenge through structural decomposition. However, they rely on predefined compositional templates, which can limit the space of expressible rules. Neither framework includes a mechanism for jointly learning rule structure and evaluating rule utility against the knowledge graph in a closed optimization loop.

2.4.2 LLM-augmented Rule Mining

A more recent line of work exploits the semantic knowledge encoded in large language models (LLMs) to guide rule discovery.

ChatRule [15] uses LLMs to mine logical rules over knowledge graphs. Rather than relying exclusively on graph structure and path statistics, ChatRule leverages the semantic understanding of LLMs to infer candidate rules and their properties from relation names, textual descriptions, and external knowledge. This is particularly useful when the graph is sparse or when the relation labels carry significant semantic information that pure graph-based mining would miss.

The strength of this approach lies in its ability to propose rules that may never appear as explicit paths in the graph but are semantically plausible. The limitation is that it depends on the quality and coverage of the underlying LLM, introduces an external component that is expensive to run and difficult to fine-tune for specific graphs, and does not include a mechanism for iteratively refining rules based on their actual predictive utility on the knowledge graph.

2.4.3 Latent-variable Rule Generation: RNNLogic

RNNLogic [17] adopts a fundamentally different strategy from the approaches above. Instead of imposing compositional structure or injecting external semantic knowledge, it treats logic rules as latent variables within a probabilistic framework and jointly trains two components: a rule generator that proposes candidate rules for each query relation, and a reasoning predictor that scores candidates based on how well they explain observed triples.

The rule generator is parameterized by a recurrent neural network (specifically, an LSTM in the original work) that produces rules as sequences of relations conditioned on the query relation. The reasoning predictor is a log-linear model that grounds the generated rules on the knowledge graph and computes scores for candidate answers. These two components form a closed loop: the generator proposes rules, the predictor evaluates them, and the feedback from evaluation is used to improve the generator. This mutual refinement is orchestrated through an Expectation-Maximization (EM) algorithm, where the E-step selects high-utility rules based on their posterior scores (called H-scores) and the M-step updates the generator to maximize the likelihood of producing those rules.

A key feature of this architecture is that it can optionally incorporate a knowledge graph embedding model (specifically RotatE) within the reasoning predictor, so that symbolic rule evidence and continuous embedding signals are combined at scoring time. The enhanced predictor variant, RNNLogic+, further integrates Principal Neighbourhood Aggregation (PNA) for richer feature aggregation, achieving competitive performance with state-of-the-art embedding models while retaining the explicit rule interface.

2.4.4 Comparison of Neurosymbolic Approaches

The three categories described above represent meaningfully different strategies for combining neural and symbolic reasoning, and each comes with characteristic strengths and trade-offs. Table 2.1 summarizes the main differences.

Compositional methods (NCRL, RLogic) address the combinatorial search problem through structural decomposition, which promotes generalization and parameter sharing. However, their predefined compositional templates constrain the expressible rule space, and they lack a mechanism for evaluating and iteratively improving rule quality against the graph. LLM-augmented methods (ChatRule) can propose semantically plausible rules that graph mining alone might miss, but they depend on an external, expensive component and do not refine rules based on actual predictive performance.

RNNLogic occupies a distinctive position by coupling rule generation with rule evaluation in a single trainable framework. The EM loop provides an explicit mechanism for iterative refinement: rules that prove useful for prediction are reinforced in the generator, while poor rules are gradually suppressed. Furthermore, the ability to integrate embedding signals within the predictor means that RNNLogic can leverage the predictive power of embedding models (which, as discussed in Section 2.1, consistently achieve strong performance) without sacrificing the interpretability of explicit rules. This combination of closed-loop optimization and embedding integration is not available in the compositional or LLM-augmented approaches.

Table 2.1: Comparison of neurosymbolic approaches for knowledge graph reasoning.

	Compositional (NCRL, RLogic)	**LLM-augmented** (ChatRule)	**Latent-variable** (RNNLogic)
Rule representation	Hierarchical compositions of atomic subrules	Rules mined via LLM semantic reasoning	Flat sequences generated by an RNN
Integration with KG	Rule grounding on graph structure	Graph structure + external LLM knowledge	Closed-loop with graph-grounded predictor and optional embeddings
Optimization	Template-based learning with shared parameters	One-shot or few-shot LLM prompting	Iterative EM between generator and predictor
Scalability	Good (parameter sharing reduces redundancy)	Limited by LLM inference cost	Good (efficient sampling and batched grounding)
Iterative refinement	No closed feedback loop	No iterative refinement	Yes (EM loop with H-score feedback)
Embedding integration	Not natively supported	Not natively supported	Built-in (RotatE in predictor, PNA in RNNLogic+)

2.4.5 Why RNNLogic for our work

Several properties of RNNLogic make it particularly well suited as the foundation for our project. First, its explicit rule interface means that every prediction can, in principle, be traced back to specific symbolic rules, which is essential for the interpretability goals of our work. Second, the EM-based training loop, while effective, represents a natural point for improvement: the E-step involves expensive full evaluation of all candidate rules, and the generator uses an LSTM whose sequential bottleneck and parameter overhead may be unnecessary for the typically short rule sequences (3–5 hops) encountered in practice. Third, the modular separation between generator and predictor allows us to modify one component without disrupting the other, facilitating controlled experiments.

These observations directly motivate our three contributions: replacing the LSTM generator with a more parameter-efficient GRU, augmenting the EM loop with a value network that pre-filters candidates before expensive evaluation (inspired by value-based RL), and integrating attention mechanisms into the rule representation to provide finer-grained insight into which parts of a rule drive predictions. The next chapter (Chapter 3) describes the RNNLogic framework in full technical detail, including the rule representation, generator architecture, predictor formulation, and the original training procedure that our methods build upon.

Chapter 3

Background

This chapter establishes the technical foundations necessary to understand the implementation and execution of the project. We detail the architecture of the RNNLogic framework, the mathematical formulation of the embeddings used, the mechanics of rule mining and generation, and the optimization strategies employed.

3.1 RNNLogic Overall Architecture

The RNNLogic framework operates as a multi-stage pipeline designed to progressively refine logical rules and enhance reasoning capabilities [17]. The overall architecture is divided into three distinct phases, where the output of one phase serves as the initialization or input for the next.

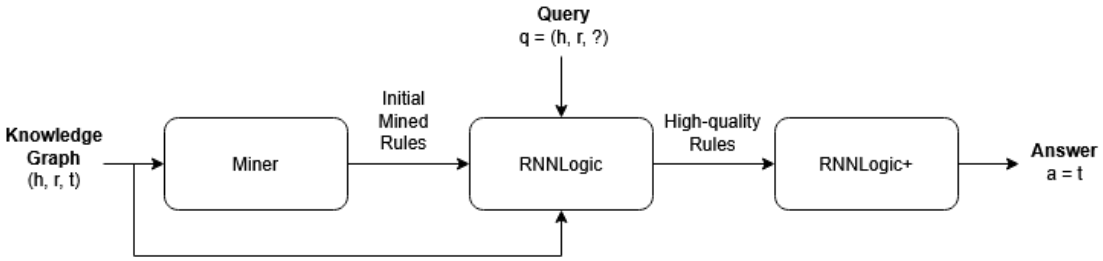

Figure 3.1: The three-phase architecture of the RNNLogic framework: (1) the Miner extracts initial path-based rules, (2) RNNLogic refines these into a generative model which outputs high-quality rules, and (3) RNNLogic+ utilizes the learned rules for enhanced reasoning to give the prediction.

The workflow proceeds as follows:

1. **Phase 1: The Miner (Initialization).** The process begins with a non-parametric rule mining module. This component exhaustively searches the knowledge graph for relational paths that connect entity pairs observed in the training data. These mined paths are converted into an initial set of logic rules, which provide a "cold start" initialization for the generative model in the next phase. This prevents the system from starting with a completely random search space.

2. **Phase 2: RNNLogic (Generative Rule Learning).** The core phase of the framework treats logic rules as latent variables. It consists of a **Rule Generator** (parameterized by a neural network) and a **Reasoning Predictor**. These two components are trained jointly using an Expectation-Maximization (EM) algorithm. The Rule Generator proposes compositional rules, while the Reasoning Predictor evaluates their validity against the ground truth. Through iterative updates, the generator learns to produce high-quality rules that effectively explain the data.

3. **Phase 3: RNNLogic+ (Enhanced Reasoning).** In the final phase, the high-quality rules learned by RNNLogic are frozen and fed into a more powerful reasoning predictor called RNNLogic+. Unlike the simple predictor used during EM training, RNNLogic+ integrates the logic rules with knowledge graph embeddings (e.g., RotatE) and advanced aggregation mechanisms (PNA). This allows the model to combine symbolic logical evidence with continuous embedding similarity for state-of-the-art prediction performance. **This phase is optional**, as the predictor trained in Phase 2 is fully capable of inference and can perform predictions using the set of high-quality rules identified by the generator.

The following sections detail the theoretical and practical implementation of each phase.

3.2 Phase 1: Initialization with the Miner

Before the rule generation process begins, it is necessary to extract an initial set of logic rules directly from the graph structure. This process is referred to as mining.

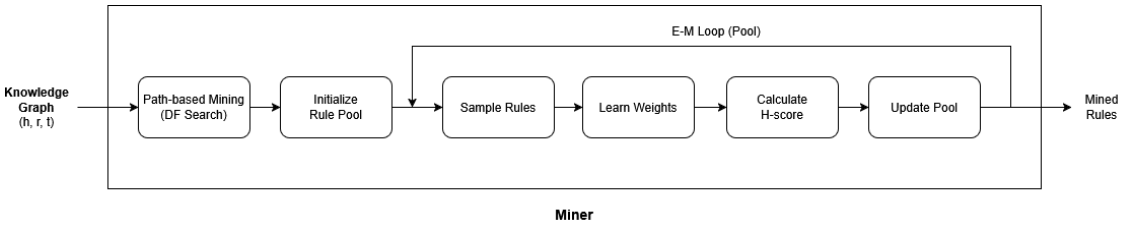

Figure 3.2: The detailed workflow of the Miner. The process begins with **Path-Based Search**, where a Depth-First Search (DFS) extracts raw logical paths from the Knowledge Graph by temporarily removing direct edges from training triplets. These candidate rules then enter an **Iterative Refinement Loop**, where rule weights are learned and H-scores are computed over multiple iterations to filter out noise. Finally, (4) the system outputs a set of mined rules to initialize Phase 2.

3.2.1 Path-based Mining

Path-based mining operates on the principle that if a relation $r(h, t)$ exists, there are likely alternative paths connecting h to t in the graph that *explain* this relation. The miner implements a depth-first search (DFS) approach that enumerates all such paths up to a specified maximum length.

For each training triplet (h, r, t), the miner temporarily removes the direct edge to prevent trivial cycles. It then searches for paths $h \xrightarrow{r_1} e_1 \xrightarrow{r_2} \dots \xrightarrow{r_L} t$. If such a path is found, it is abstracted into a logic rule:

$$r(h, t) \leftarrow r_1(h, Z_1) \wedge r_2(Z_1, Z_2) \wedge \cdots \wedge r_L(Z_{L-1}, t) \tag{3.1}$$

The mining process consists of four main steps:

1. **Graph Loading and Preprocessing:** The miner first loads the knowledge graph and constructs adjacency lists for efficient traversal. For each entity e and relation

r, it maintains a list of all entities reachable from e via r. This preprocessing enables fast neighbor lookups during the path search.

2. **Rule Discovery via Depth-First Search:** For each training triplet (h, r, t), the miner searches for alternative paths from h to t that do not use the direct edge. The triplet (h, r, t) being examined is temporarily excluded from the graph to prevent discovering the trivial single-hop rule $r \leftarrow r$. Starting from h with goal t, the algorithm explores all possible paths up to a maximum length L. Rules discovered across all training triplets are collected into relation-specific sets, with duplicates automatically removed.

   Concrete Example. Consider mining rules for the triplet (`person15`, `term7`, `person55`) with `max_length=2`:

   (a) The search begins at `person15` with goal `person55`

   (b) The direct edge `person15` $\xrightarrow{\texttt{term7}}$ `person55` is removed

   (c) The algorithm explores alternative paths:
   - Path: `person15` $\xrightarrow{\texttt{term16}}$ `person103` $\xrightarrow{\texttt{term0}}$ `person23` $\rightarrow$ max depth reached, goal not found
   - Path: `person15` $\xrightarrow{\texttt{term18}}$ `person43` $\xrightarrow{\texttt{term1}}$ `person55` $\rightarrow$ **goal reached!**

   (d) The successful path yields the rule:

$$\texttt{term7}(X, Y) \leftarrow \texttt{term18}(X, Z) \wedge \texttt{term1}(Z, Y)$$

   This rule captures the pattern: "if X has `term18` relation with Z, and Z has `term1` relation with Y, then X likely has `term7` relation with Y."

3. **Rule Weight Learning:** After discovery, the miner learns a scalar weight $\psi(\text{rule})$ for each rule through iterative optimization. For a query $(h, r, ?)$ with candidate answers A, the score for each candidate $e \in A$ is computed as:

$$\text{score}(e) = \sum_{\text{rule} \in z_r} \psi(\text{rule}) \cdot |\mathcal{P}(h, \text{rule}, e)| \tag{3.2}$$

   where z_r is the set of rules with head relation r, and $|\mathcal{P}(h, \text{rule}, e)|$ counts the number of grounding paths from h to e following the rule body.

   Probabilities are obtained via softmax normalization:

$$p(e \mid h, r, z) = \frac{\exp(\text{score}(e)/\tau)}{\sum_{e' \in A} \exp(\text{score}(e')/\tau)} \tag{3.3}$$

   where τ is a temperature parameter. Rule weights are updated using the Adam optimizer to maximize the likelihood of correct answers.

4. **H-Score Computation and Rule Selection:** The quality of each rule is assessed using the H-score, which measures how much a rule contributes to correct predictions relative to incorrect ones:

$$H(\text{rule}) = \text{score}(t \mid \text{rule}) - \frac{1}{|A|} \sum_{e \in A} \text{score}(e \mid \text{rule}) \tag{3.4}$$

where t is the correct tail entity. A positive H-score indicates the rule is more predictive of correct answers than random candidates. The top-k rules by H-score are selected for each triplet during the E-step of the EM algorithm.

3.2.2 Implementation Details and Key Parameters

The Miner module is implemented in C++ to handle the high computational cost of graph traversal. Its performance and the quality of the mined rules are governed by a specific set of hyperparameters.

The mining process involves path finding, controlled by `max-length` and `threads`, and rule scoring, controlled by optimization parameters like `lr` (learning rate) and `wd` (weight decay). The parameter `max-length` is particularly crucial. While longer rules (e.g., length 3 or 4) can capture more complex reasoning chains, they increase the search space exponentially. To manage this, the `top-n` and `top-k` parameters are used to prune the rule set, ensuring only the most statistically significant rules are passed to the next phase.

Parameter	Description	Typical Values
`max-length`	Maximum rule body length	3
`iterations`	Number of weight learning iterations	1–10
`top-k`	Rules selected per triplet in H-score computation	10
`top-n`	Rules sampled per relation during learning	100–2000
`top-n-out`	Number of final rules output (0 = unlimited)	100–500
`lr`	Learning rate for weight updates	0.01
`wd`	Weight decay regularization	0.0005
`temp`	Temperature for softmax normalization	100
`threads`	Number of parallel threads	40

Table 3.1: Key parameters for the RNNLogic miner.

3.2.3 Efficient Grounding with Dynamic Programming

To evaluate the mined rules, the miner must compute the number of paths connecting a head entity to all potential tail entities following a specific rule body. Naive graph traversal is computationally expensive. Instead, the miner uses dynamic programming to compute all destination counts simultaneously.

Algorithm 1 Dynamic Programming Path Counter

```
procedure RuleDestination(startEntity, rule, excludedTriplet)
    paths[0][startEntity] ← 1                          ▷ Start: 1 path of length 0
    for each rule step k = 0 to |rule| − 1 do
        relation ← rule[k]
        for each entity e where paths[k][e] > 0 do
            for each neighbor n via relation from e do
                if (e, relation, n) ≠ excludedTriplet then
                    paths[k + 1][n] ← paths[k + 1][n] + paths[k][e]
                end if
            end for
        end for
    end for
    return paths[|rule|]                               ▷ All destination path counts
end procedure
```

This computes path counts to all reachable entities in a single forward pass, enabling efficient batch scoring.

3.2.4 Role in RNNLogic

In the RNNLogic framework, path-based mining serves two purposes:

1. **Cold Start Initialization:** The mined rules provide an initial pool for the rule generator, preventing exploration of meaningless rules during early training.

2. **Training Signal Generation:** Rules scored by H-values become weighted training examples for the rule generator, teaching it which patterns are effective for reasoning.

3.2.5 Limitations

Despite its effectiveness, path-based mining has several limitations:

- **Exponential Complexity:** Search space grows as $O(|R|^L \cdot d^L)$ where $|R|$ is the number of relations, L is maximum length, and d is average node degree.

- **Path Dependency:** Cannot discover semantic patterns (e.g., symmetry) not explicitly instantiated as paths in the training data.

- **Redundancy:** May produce rules capturing similar patterns, requiring subsequent filtering.

3.3 Phase 2: Generative Rule Learning with RNNLogic

RNNLogic (Recurrent Neural Network Logic) is a probabilistic framework designed to solve the Knowledge Graph reasoning problem by treating logic rules as latent variables [17]. The architecture is predicated on the idea that identifying the correct logic rules

to answer a query is a generative process, while applying those rules to the graph is a reasoning process.

Formally, given a knowledge graph $\mathcal{G}$ (a set of triplets) and a query $q = (h, r, ?)$, we aim to compute the probability of a candidate answer a, denoted as $p(a \mid \mathcal{G}, q)$. RNNLogic decomposes this probability by introducing a latent variable z, which represents a set of logic rules. The joint probability is marginalized over these rules:

$$p_{w,\theta}(a \mid \mathcal{G}, q) = \sum_z p_w(a \mid \mathcal{G}, q, z) p_\theta(z \mid q) = \mathbb{E}_{p_\theta(z|q)}[p_w(a \mid \mathcal{G}, q, z)]$$

There are two primary components in the architecture: the rule generator and the reasoning predictor. The system forms a closed loop: the rule generator proposes rules, the reasoning predictor evaluates them on the data, and the feedback from the reasoning predictor is used to refine the rule generator. This mutual enhancement is orchestrated via an Expectation-Maximization (EM) algorithm.

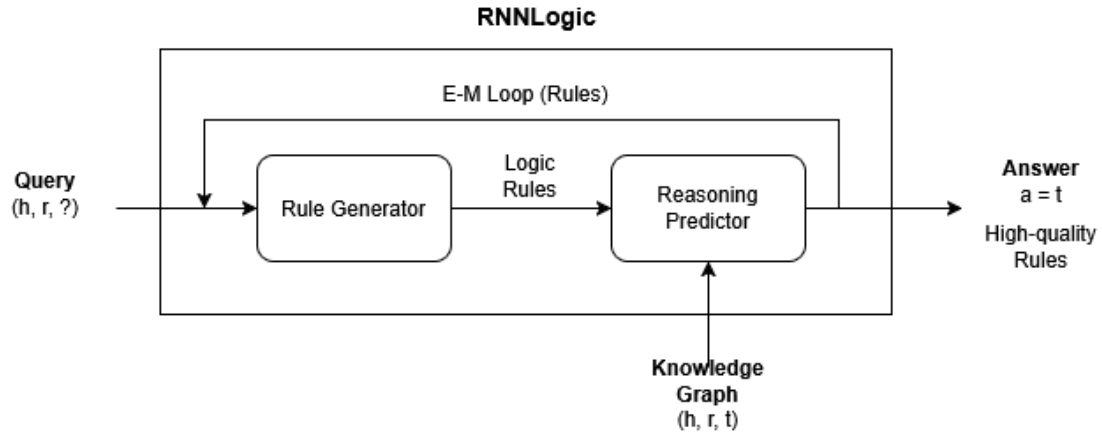

Figure 3.3: The detailed workflow of RNNLogic. The process is initiated by a query $(h, r, ?)$. The **Rule Generator** (p_θ, built with LSTM) conditions on the query relation r to propose a set of latent logic rules z. These rules are fed into the **Reasoning Predictor** (p_w), which grounds them on the knowledge graph $\mathcal{G}$ to compute scores and probabilities for candidate answers. The two components are jointly optimized via an **Expectation-Maximization (EM) loop**. In the E-step, the predictor evaluates rules to select a high-quality subset (z^I) based on H-scores. In the M-step, the rule generator updates its parameters θ to maximize the likelihood of generating these selected rules.

3.3.1 The Rule Generator

The Rule Generator $p_\theta(z \mid q)$ defines a prior distribution over logic rules, treating them as latent variables. We employ a Long Short-Term Memory (LSTM) network to generate rules as sequences of relations, conditioned explicitly on the query relation r.

The choice of LSTM is critical for modeling the sequential dependencies inherent in logic rules. Standard Recurrent Neural Networks (RNNs) often suffer from the vanishing gradient problem, making it hard to learn logic chains with multiple hops. LSTMs mitigate this through sophisticated gating mechanisms (input, output, and forget gates) that regulate the flow of information, allowing the model to preserve gradients and capture long-range dependencies effectively [9].

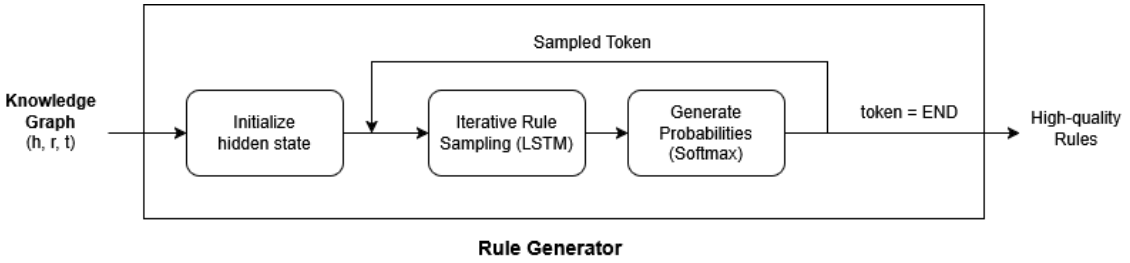

Figure 3.4: The detailed workflow of the Rule Generator. The process begins with **Initialization**, where the query relation r is embedded into a vector $\mathbf{v}_r$ to initialize the LSTM hidden state. The system then enters a **Sequential Generation Loop**, where at each time step t, the LSTM conditions on both the previous relation embedding $\mathbf{v}_{r_{t-1}}$ and the original query embedding $\mathbf{v}_r$ (via concatenation) to maintain context. A softmax layer projects the hidden state to a probability distribution over relations, from which the next relation r_t is sampled. This loop continues until an END token is generated, producing complete logic rules z.

The generation process for a rule body $r_1 \wedge \cdots \wedge r_L$ is modeled sequentially. The LSTM is initialized with the embedding of the query relation, $\mathbf{v}_r$. Crucially, to ensure the generated rule remains relevant to the query throughout the sequence, the input to the LSTM at each step t is the concatenation of the previous relation's embedding $\mathbf{v}_{r_{t-1}}$ and the query relation's embedding $\mathbf{v}_r$:

$$\mathbf{h}_t = \text{LSTM}(\mathbf{h}_{t-1}, [\mathbf{v}_{r_{t-1}}; \mathbf{v}_r]) \tag{3.5}$$

The probability of selecting the next relation r_t is computed via a softmax layer over the relation vocabulary:

$$p(r_t \mid r_{<t}, r) = \text{Softmax}(W\mathbf{h}_t + b) \tag{3.6}$$

The probability of a complete rule z is the product of the probabilities of its constituent relations. This architecture allows the model to explore the combinatorial space of logic rules continuously and propose novel patterns that may not exist as explicit paths in the incomplete training graph.

3.3.2 The Reasoning Predictor

The Reasoning Predictor $p_w(a \mid \mathcal{G}, q, z)$ evaluates the utility of the rules generated by the prior. It functions as a log-linear model that scores candidate answers based on the groundings of the provided rules z.

For a query $q = (h, r, ?)$, the predictor calculates a scalar score for each candidate entity e by aggregating evidence across all rules:

$$\text{score}_w(e) = \sum_{\text{rule} \in z} \psi_w(\text{rule}) \cdot \left(\sum_{\text{path} \in \mathcal{P}(h, \text{rule}, e)} \phi_w(\text{path}) \right) \tag{3.7}$$

where:

- $\psi_w(\text{rule})$ is a learnable scalar weight associated with each specific rule, initialized to reflect the rule's empirical precision or set randomly. This weight allows the predictor to downweight irrelevant rules proposed by the generator.

- $\phi_w(\text{path})$ is a score for the specific path instance. In the basic configuration, $\phi_w = 1$ (counting paths). In the embedding-enhanced configuration, this term is derived from the RotatE score of the path, providing a measure of geometric consistency.

The final probability of an answer a is obtained by normalizing these scores over all candidate entities $\mathcal{A}$ using the softmax function:

$$p_w(a \mid \mathcal{G}, q, z) = \frac{\exp(\text{score}_w(a))}{\sum_{e' \in \mathcal{A}} \exp(\text{score}_w(e'))} \tag{3.8}$$

3.3.3 Optimization: The EM Algorithm

We jointly train the generator and predictor using an Expectation-Maximization (EM) framework to maximize the log-likelihood of the training data.

E-step (Rule Selection). We generate a set of candidate rules $\hat{z}$ from the generator. We then identify a subset of "high-quality" rules z^I by computing the **H-score** for each rule:

$$H(\text{rule}) = \text{score}_w(t \mid \text{rule}) - \frac{1}{|A|} \sum_{e \in A} \text{score}_w(e \mid \text{rule}) + \log p_\theta(\text{rule} \mid r) \tag{3.9}$$

This score balances the rule's empirical accuracy on the current graph (likelihood) with its plausibility under the current generator (prior).

M-step (Generator Update). The Rule Generator is updated to maximize the likelihood of generating the high-quality rules selected in the E-step:

$$\max_\theta \sum_{\text{rule} \in z^I} \log p_\theta(\text{rule} \mid r) \tag{3.10}$$

This closes the loop: the predictor identifies which rules work, and the generator learns to produce them.

3.3.4 Implementation Details and Key Parameters

The RNNLogic module is implemented in Python using the PyTorch framework to leverage GPU acceleration for neural network training. Its performance relies heavily on the capacity of the neural networks and the breadth of the rule search space managed during the EM optimization.

The generative learning process is controlled by parameters governing the Rule Generator (e.g., `hidden-dim`, `max-length`) and the Reasoning Predictor. Crucially, the `num-sample` and `top-k` parameters define the scope of the E-step, effectively balancing the exploration of novel logical patterns against the exploitation of high-quality rules confirmed by the predictor. The learning rates for the two components (`gen-lr`, `pred-lr`) are distinct and must be carefully tuned to ensure the stable convergence of the alternating optimization loop.

Parameter	Description	Typical Values
`hidden-dim`	Hidden state size of the LSTM generator	256
`input-dim`	Input embedding size for relations	512
`max-length`	Maximum generated rule length	3
`num-sample`	Rules sampled per query in E-step (N)	1000
`top-k`	High-quality rules selected for M-step (K)	300
`gen-lr`	Learning rate for Rule Generator	1×10^{-3}
`pred-lr`	Learning rate for Reasoning Predictor	5×10^{-5}
`batch-size`	Number of queries per training batch	16–32
`iterations`	Total training steps (EM iterations)	5-10

Table 3.2: Key parameters for the RNNLogic generative learning phase.

3.4 Phase 3: Enhanced Reasoning (RNNLogic+)

Once the high-quality rules are learned, phase 3 focuses on maximizing predictive performance. RNNLogic+ is a standalone predictor that combines the symbolic strengths of phase 2 with the expressive power of knowledge graph embeddings.

3.4.1 Knowledge Graph Embeddings (RotatE)

RNNLogic+ incorporates the RotatE model [22], which maps entities and relations to the complex vector space $\mathbb{C}^d$. Relations are modeled as rotations: given a triplet (h, r, t), we expect $\mathbf{h} \circ \mathbf{r} \approx \mathbf{t}$.

This geometric interpretation allows the model to capture logical patterns such as symmetry ($\theta_r \in \{0, \pi\}$), inversion ($\theta_{r_1} + \theta_{r_2} = 0$), and composition by simple angle addition. In RNNLogic+, this rotational distance is used to score the "soundness" of reasoning paths:

$$\mathrm{KGE}(h, r, e) = \|\mathbf{h} \circ \mathbf{r}_1 \circ \cdots \circ \mathbf{r}_L - \mathbf{e}\| \tag{3.11}$$

3.4.2 Predictor+ Architecture with PNA

Unlike the simple summation used in Phase 2, RNNLogic+ employs Principal Neighbourhood Aggregation (PNA) to fuse rule counts and embedding scores [7].

PNA computes a rich feature set by applying multiple aggregators and scalers to the input signals. In the original formulation, 12 features are used (4 aggregators $\times$ 3 scalers):

- **Aggregators:** mean, max, min, standard deviation.
- **Scalers:** identity, amplification ($\times \log(\text{degree})$), attenuation ($/ \log(\text{degree})$).

This yields a detailed feature vector that allows the MLP to distinguish between different evidence patterns. For a candidate e, the final score is given by:

$$\mathrm{score}(e) = \mathrm{MLP}\left(\mathrm{AGG}\left(\{(\mathbf{v}_{\mathrm{rule}}, |\mathcal{P}(h, \mathrm{rule}, e)|)\}_{\mathrm{rule} \in z^I}\right)\right) + \eta \cdot \mathrm{KGE}(h, r, e) \tag{3.12}$$

Here, η controls the weight of the direct RotatE score, allowing the model to dynamically weight logical evidence based on its statistical properties.

3.5 Our Contribution

While the RNNLogic framework comprises three distinct phases, our work primarily focuses on the analysis of **Phase 2: Generative Rule Learning with RNNLogic**. Although we delved into the the other two modules initially, our substantial contributions lie in re-engineering the core generative mechanism and its optimization strategy, which will be discussed further in the next chapter.

Chapter 4

Our Methodology and Approach

This chapter presents a targeted re-engineering of the RNNLogic framework [17] for logic rule learning on knowledge graphs. While RNNLogic achieves strong performance via an Expectation–Maximization (EM) mechanism between a rule generator and a reasoning predictor, it exhibits two main limitations: (1) the LSTM-based rule generator introduces sequential bottlenecks and parameter inefficiency on sparse rule data, and (2) the EM loop becomes computationally expensive by evaluating all generated candidates with the reasoning predictor. In addition, although RNNLogic learns latent rules, it offers limited tools to inspect which parts of a rule contribute most to a prediction, restricting its interpretability. We address these issues through three complementary contributions: a GRU-based rule generator, value-guided EM algorithm, and an attention-driven mechanism for fine-grained, relation-level explainability.

4.1 Overall Methodology

Our methodology introduces three integrated modifications to the RNNLogic pipeline, while preserving the original path-based initialization and RNNLogic+ reasoning components to enable direct comparison:

1. **GRU-based Rule Generator:** Replace the LSTM with a parameter-efficient Gated Recurrent Unit that maintains temporal modeling while reducing overfitting on sparse mined rules.

2. **Value-guided EM Optimization:** Augment the EM loop with a lightweight Value Network that predicts H-scores [17] to pre-filter large candidate sets before expensive reasoning predictor evaluation.

3. **Attention-based Interpretability:** Integrate attention mechanisms to quantify edge importance in generated rules, and expose these attributions through an interactive visualization interface.

Complete hyperparameters are provided in Appendix D.

4.2 Rule Initialization

For the initialization phase, we utilize the default path-based miner to generate an initial pool of rules for all datasets. These mined rules initialize our rule generator and ensure a consistent baseline comparison for our changes in RNNLogic.

4.3 GRU-Based Rule Generator

The **Gated Recurrent Unit (GRU)** is a simplified variant of the recurrent neural network that addresses the vanishing gradient problem without the structural complexity of the LSTM [6]. It retains the ability to model temporal dependencies but merges the internal cell state and hidden state into a single unified representation. By reducing the number of gating operations, it offers a more lightweight and faster alternative for sequence generation, which is particularly beneficial for logic rule learning where the training data (mined rules) can be sparse or noisy. The modified rule generator is shown in Figure 4.1,

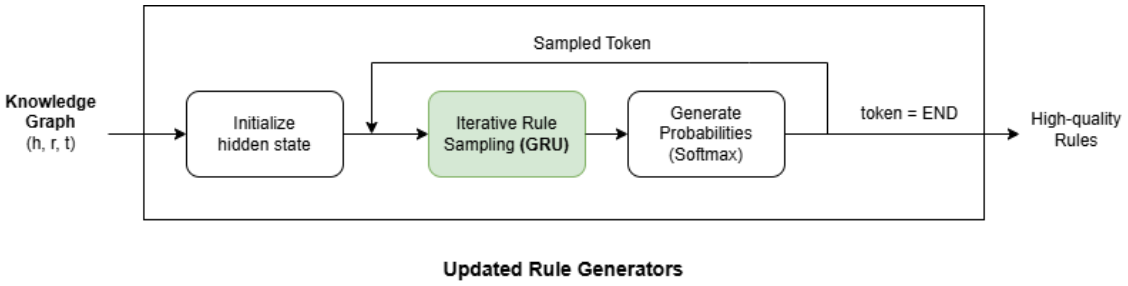

Figure 4.1: GRU-based rule generator architecture (highlighted in green) replacing the LSTM baseline.

Architectural Mechanics

The GRU simplifies the LSTM's three-gate structure into two gates: the **reset gate** (r_t) which determines how much of the past information to forget, and the **update gate** (z_t) which determines how much to carry forward. The mathematical formulation follows the implementation by Cho et al. [5] and can be summarized into three steps:

1. **Gates.** The gates are computed using sigmoid activations based on the current input x_t and the previous hidden state h_{t-1}:

$$z_t = \sigma(W_z x_t + U_z h_{t-1}) \tag{4.1}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1}) \tag{4.2}$$

   The update gate z_t acts as a "slider." If $z_t \approx 1$, the unit accepts the new candidate state. If $z_t \approx 0$, it ignores the new input and retains the old state. The reset gate r_t allows the model to "reset" its memory; if $r_t \approx 0$, the unit ignores the previous state h_{t-1} when computing the new candidate.

2. **Candidate Activation.** The candidate activation represents the proposed new memory content. Crucially, the reset gate r_t is applied to the previous hidden state via element-wise multiplication ($\odot$) before the linear transformation:

$$\tilde{h}_t = \tanh(W x_t + U(r_t \odot h_{t-1})) \tag{4.3}$$

   By multiplying $r_t \odot h_{t-1}$, the model effectively drops information from the past context that is irrelevant to the current prediction before generating the new candidate vector. This allows the GRU to handle "scene changes" in logic rules efficiently.

3. **Final Hidden State.** The final hidden state at time t is a linear interpolation between the previous activation h_{t-1} and the candidate activation $\tilde{h}_t$:

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{4.4}$$

This linear sum allows the GRU to expose the whole state at each step while adaptively controlling the decay of old information. Unlike the LSTM, it does not possess a separate memory cell, making it more parameter-efficient.

Theoretical Advantages and Hypothesized Use Cases

The GRU offers three key theoretical advantages over LSTM for our specific use case of logic rule generation on knowledge graphs:

1. **Parameter Efficiency:** By eliminating the output gate and separate memory cell, GRUs typically require fewer parameters than LSTMs of the same hidden dimension. This reduction is particularly beneficial for logic rule generation, where training data consists of sparse, noisy mined rules from path-based initialization. Additionally, the simpler gradient flow can often lead to faster convergence during the training phases. We expect the GRU to excel on smaller datasets or in scenarios where training speed is a primary concern.

2. **Robustness to Short Sequences:** Rule sequences in knowledge graph reasoning tasks are typically short, constrained to 3-5 hops. While LSTMs theoretically excel at capturing long-range dependencies, their complex gating can overfit on these brief sequences. The GRU's simpler two-gate mechanism provides sufficient expressive power for short-term dependencies while avoiding unnecessary representational capacity.

3. **Reduced Overfitting on Small Datasets:** RNNLogic operates on relatively small benchmark datasets where mined rules constitute limited training examples. The GRU's parameter efficiency and smoother gradient flow reduce overfitting risk compared to LSTM, while maintaining the recurrent inductive bias essential for sequential logical reasoning. When paired with the value network, the GRU's efficiency enables rapid iteration over candidate sets.

We hypothesize that these properties make GRU particularly well-suited as our primary architecture, serving as a lightweight yet robust baseline that prioritizes generalization over the maximum representational capacity offered by LSTM.

4.4 Value-guided EM Optimization

The standard RNNLogic framework relies on an Expectation-Maximization (EM) loop where the rule generator creates candidates, and the reasoning predictor evaluates them to compute H-scores. This process can be computationally expensive because training the reasoning predictor for every batch of generated rules is slow. To address this, we introduce a **Value Network** that learns to estimate the H-scores directly from rule sequences,

enabling efficient pre-filtering of candidates before reasoning predictor evaluation. While inspired by reinforcement learning concepts, particularly the value networks from deep RL [21], this is fundamentally supervised learning where the value network regresses true H-scores from the knowledge graph. This allows us to sample a large number of candidates and filter them efficiently, passing only the most promising rules to the expensive evaluation step.

The framework can be configured to generate N candidate rules and select the top-K (with $N \gg K$), theoretically accelerating training by reducing predictor evaluations by a factor of N/K. However, actual speedups depend on the chosen K and hardware constraints. In our experiments, we fixed N = 4000 and K = 1500 across all tests to ensure fair comparison.

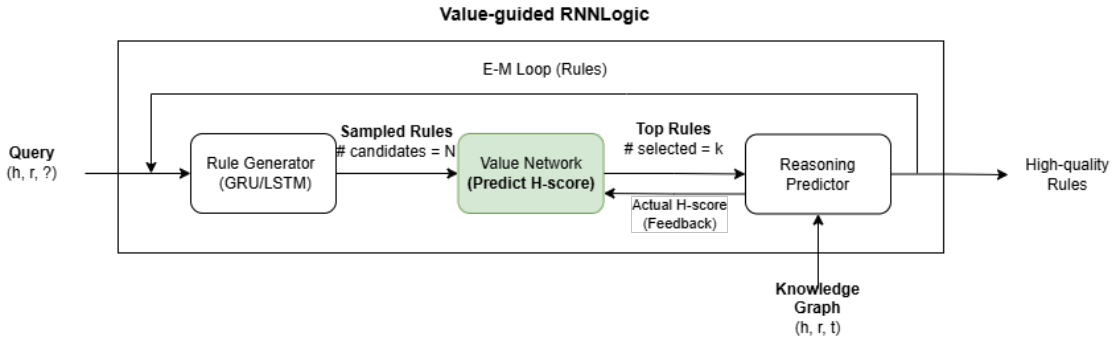

Figure 4.2: Value-guided optimization pipeline with Value Network (green) for H-score prediction and rule filtering.

RL-inspired Architecture

The architecture consists of three interacting modules, each with specialized functionality as illustrated in Figure 4.2. The different modules are described below, along with their corresponding RL components to help contextualize the concepts.

- **Rule Generator (Policy $\pi(a|s)$):** A recurrent sequence model (GRU or LSTM) that autoregressively generates candidate logic rules conditioned on a target query relation r_{target}. In RL terms, the state $s_t = [r_{\text{head}}, r_1, \ldots, r_{t-1}; r_{\text{target}}]$ represents the partial rule history plus target context, and the action $a_t = r_t$ is the next relation to append. The generator thus implements the policy $\pi(a_t|s_t) = P(r_t|r_{\text{head}}, r_1, \ldots, r_{t-1}; r_{\text{target}})$ which stochastically selects the next step in rule construction, analogous to how an RL policy selects actions to maximize expected future reward (in this case, H-score) in a sequential decision process.

- **Value Network (Critic $V(s)$):** A dedicated neural network that takes a complete rule sequences $[r_{\text{head}}, r_1, \ldots, r_L]$ as input and regresses a scalar prediction $\hat{V}(r)$ approximating the H-score. The architecture comprises of:

  1. **Relation Embedding Layer:** Maps each discrete relation ID to a dense d-dimensional vector.

2. **Bi-directional GRU Encoder:** Processes the full sequence bidirectionally to capture both forward logical flow and backward context.
3. **MLP Regression Head:** Projects concatenated states through two fully-connected layers with ReLU activation to produce the final scalar representing the predicted H-score.

The bi-directional design ensures the network understands complete logical chains rather than prefix information.

- **Reasoning Predictor (Environment):** The original unchanged component that computes ground-truth H-scores $H(r)$ for evaluation rules. For a candidate rule r, it instantiates all logical groundings across the graph and measures the rule's H-score using the original graph. This serves as the supervisory "reward signal" for value network training.

Updated EM Procedure

The algorithm proceeds as follows:

1. **Large-Scale Generation:** The rule generator (GRU/LSTM) samples a large batch of N candidate rules for the target relation. These represent diverse exploration of the possible rule space, exceeding the batch sizes used in standard RNNLogic.
2. **Value-Based Filtering:** The value network evaluates all N candidates, predicting an H-score estimate for each complete rule. We select only the top-K most promising rules based on these predictions. This filtering step is expected to perform faster than reasoning predictor evaluation.
3. **Ground-Truth Evaluation:** Only the filtered top-K rules are passed to the reasoning predictor, which computes their true H-scores $H(r)$ by evaluating prediction performance across all logical rules in the validation knowledge graph.
4. **Value Network Update:** The value network is trained as a regressor using the top-K rules as training examples. It minimizes the mean squared error between its predictions and the true H-scores computed in step 3, learning to better approximate the predictor's quality computations.
5. **Generator Update:** Finally, the rule generator performs the standard RNNLogic M-step, but now trained exclusively on the high-quality rules from. These rules are weighted by their posterior probability (prior from generator + likelihood from H-scores), ensuring the generator learns to produce increasingly better candidates over iterations.

Theoretical Advantages and Hypothesized Use Cases

Our value-guided optimization offers three key theoretical advantages over standard RNN-Logic EM as shown below. Initially, training may be slower without value predictions, but the reasoning predictor rapidly provides true H-score supervision, enabling quick value network convergence.

1. **Quality-Driven Learning:** Standard EM treats all generated rules equally regardless of quality during the prediction. We prioritize high-H-score rules for generator training, creating a feedback loop where the generator learns from superior examples earlier, accelerating convergence to more optimal rule sets.

2. **Larger Rule Space Exploration:** Our value network enables generating thousands of candidates while evaluating only a part of those with the highest predicted h-scores.

We hypothesize these properties yield better performance, faster convergence, and more robust rule sets than standard EM optimization. To isolate the impact of RL optimization from architectural changes, **we evaluate both LSTM with Value Network and GRU with Value Network configurations**, allowing us to decouple gains from the generator architecture versus gains from the optimization strategy.

4.5 Attention-Based Rule Interpretability

RNNLogic assigns each rule a single scalar score (the H-score), which reflects its overall utility for prediction. While the rules themselves are human-readable sequences of relations, this score provides no information about which relations within the rule are responsible for reaching the correct answer. A three-hop rule like *prevents* → *complicates* → *result_ of* → *affects* might owe its success entirely to the *complicates* step, or its value might be distributed evenly across all hops. Without a mechanism to distinguish these cases, interpretability remains at the level of whole-rule attribution rather than per-edge attribution.

We address this gap through two complementary components: an attention mechanism integrated into the rule generator that captures the model's internal view of edge importance, and a grounding-based analysis pipeline that independently measures the empirical contribution of each hop on the knowledge graph. Together, these components provide both model-level and data-level explainability, and we expose both through an interactive dashboard.

4.5.1 Attention Mechanism in the Rule Generator

We augment the GRU-based rule generator (Section 4.3) with a `nn.MultiheadAttention` layer from PyTorch [25]. During rule generation, the GRU produces a hidden state h_t at each time step t as it processes the sequence of relation tokens $[r_{\text{head}}, r_1, r_2, \ldots, r_L]$. The attention layer takes these hidden states and computes scaled dot-product attention across all positions in the rule:

$$\alpha_{ij} = \text{softmax}\left(\frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}\right), \quad \text{context}_i = \sum_j \alpha_{ij}\, \mathbf{v}_j, \tag{4.5}$$

where $\mathbf{q}_i$ is the query vector derived from the current GRU state, $\mathbf{k}_j$ and $\mathbf{v}_j$ are keys and values derived from all past relation embeddings, and d_k is the key dimension. We use multi-head attention with 4 heads, each capturing a different aspect of the positional and semantic relationships between rule edges.

The resulting attention weights α_{ij} serve as per-edge importance scores: for a given rule, α_i indicates how much the model attends to position i when generating the subsequent positions. We store these weights for all high-quality rules during inference, making them available for post-hoc analysis. While implemented on the GRU generator, this attention layer is architecture-agnostic and can be applied to any recurrent generator, including the original LSTM.

The attention mechanism sits between the generator's hidden state computation and the output softmax. It does not alter the rule generation process itself (the GRU still produces the same sequence of relation tokens), but it provides an additional read-out signal that reveals the generator's internal focus at each step. This design choice ensures that the attention layer acts purely as an interpretability probe without affecting the predictive pipeline.

4.5.2 Grounding-Based Link Contribution Analysis

The attention weights described above reflect what the model thinks is important. To measure what actually matters for prediction, we developed a complementary grounding-based analysis that is entirely independent of the model's internal representations.

The analysis works as follows. For a rule $[r_{\text{head}}, r_1, r_2, \ldots, r_L]$ and a test query $(h, r_{\text{head}}, ?)$ with correct answer t, we walk the knowledge graph step by step along the rule body. At each hop i, we track which entities are reachable and whether the correct answer t remains in the reachable set. From this, we compute three metrics at each position:

1. **Precision** at position i: the fraction of test queries where the correct answer is still reachable after following the chain up to hop i. This measures how well the partial rule path preserves the correct answer through progressive filtering of the entity space.

2. **Coverage** at position i: the fraction of test queries for this relation that have any valid path through the knowledge graph at hop i. Some paths may dead-end because the required relation does not exist for any reachable entity, so coverage tracks how broadly the rule applies.

3. **Contribution** at position i: defined as $\text{precision}_i \times \text{coverage}_i$, capturing the overall empirical importance of that hop. A hop can only contribute meaningfully if it both preserves the correct answer (high precision) and applies to many queries (high coverage).

This analysis is implemented in a standalone script (`analyze_link_contributions.py`) that takes the knowledge graph data and a set of rules as input, and produces per-link metric files that feed into the dashboard. The script constructs an adjacency structure from the training triples, groups test triples by relation, and for each rule performs step-by-step grounding across all relevant test queries using a dynamic-programming path counter. It outputs two CSV files: `per_link_metrics.csv` with per-position metrics for every rule, and `per_query_top_rules.csv` with the top-10 rules ranked by H-score for each query relation, along with their actual precision and coverage on the test set.

The distinction between the attention-derived importance (model's view) and the grounding-derived importance (empirical ground truth) is central to our explainability approach. When the two views align — the model attends to hops that are empirically important — we gain confidence that the model has learned meaningful reasoning patterns. When they diverge, this signals potential issues: the model may be relying on structurally plausible but empirically ineffective hops, or the H-score may not faithfully reflect actual predictive quality.

4.5.3 Interactive Explainability Dashboard

To make these analyses accessible and actionable, we developed an interactive Streamlit dashboard (Figure 4.3) that integrates both the attention-level and grounding-level views. The dashboard takes as input the two CSV files produced by the grounding analysis, optionally augmented with the knowledge graph triples for subgraph visualization. It is organized into four tabs:

1. **Overview and Rules.** Displays dataset-level summary statistics: the distribution of actual precision across all rules, the proportion of zero-precision rules (sparsity ratio), and the top-performing rules ranked by actual precision or H-score. For a selected rule, it renders the rule as a directed path graph with edges annotated by per-link metrics (precision, coverage, contribution), providing an immediate visual summary of where in the chain the predictive signal originates.

2. **Positions and Relations.** Shows aggregated patterns: a dual-axis chart of average contribution by rule position revealing whether early or late hops carry more predictive weight, and a heatmap of contribution (or precision, or coverage) broken down by relation type and position. This view enables identification of relations that are structurally important for reasoning, regardless of which specific rule they appear in.

3. **Rule Explorer and Knowledge Graph.** Provides a filterable, searchable table of all rules with their metrics, supporting CSV export. For a selected rule, the dashboard performs a depth-first search on the uploaded knowledge graph triples to find a concrete grounding path — an actual sequence of entities connected by the rule's relations. This path is rendered as a highlighted subgraph within the local neighbourhood of the knowledge graph, with the body path shown in teal and the direct query-relation edge (if it exists) shown in amber. This visualization makes the abstract rule concrete: the user can see exactly which real entities the rule chains through.

4. **Diagnostics.** Compares model-assigned scores (H-scores) against actual grounding precision through a calibration scatter plot, a rank-error histogram identifying the most overvalued and undervalued rules, and a side-by-side comparison of the model's top-10 ranking versus the ground-truth top-10.

The dashboard is built with Streamlit, Plotly, and NetworkX, and is designed as a tool for both research analysis and live demonstration. All visualizations are interactive

(hoverable, zoomable) and update in response to filter changes in the sidebar, where the user can restrict the view to specific query relations, minimum precision thresholds, or minimum coverage levels.

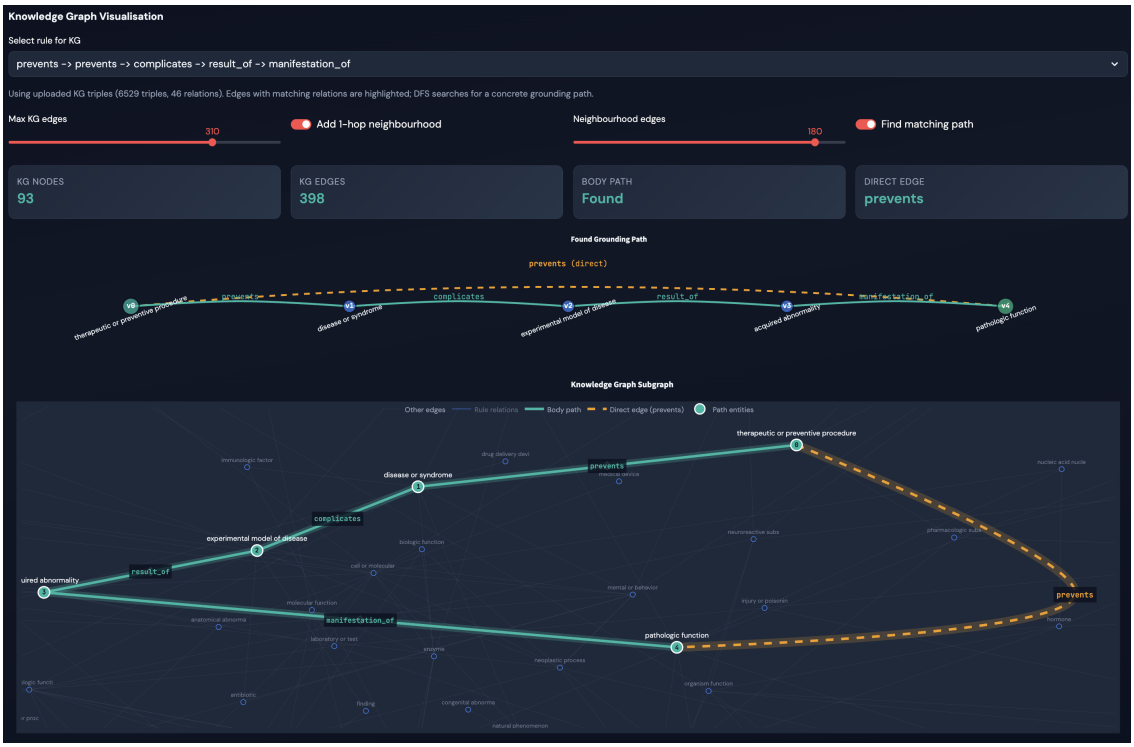

Figure 4.3: Knowledge graph visualisation from the Rule Grounding Analyzer dashboard on UMLS. Top: a concrete grounding path found by DFS for the rule *prevents* → *prevents* → *complicates* → *result_of* → *manifestation_of*, showing real biomedical entities connected by the rule's relations. Bottom: the local KG subgraph with the body path (teal) and the direct *prevents* edge (amber dashed) between the start and end entities.

4.6 Exploratory Approaches

During the development of our final pipeline, we explored several alternative approaches including semantic rule mining, biased random walk initialization, rule clustering, top-N selection, and variational inference. While these provided valuable insights into system bottlenecks and data characteristics, these approaches did not yield performance improvements over our primary contributions and were excluded from the final pipeline. Full details of these explorations are provided in Appendix B.

4.7 Summary and Experimental Roadmap

In summary, this chapter has presented a comprehensive re-engineering of the RNNLogic framework. We have addressed the limitations of the standard LSTM rule generator by introducing the GRU architecture, which offers parameter efficiency and faster convergence. Furthermore, we have augmented the standard Expectation-Maximization loop

with a value-guided rule generation framework that utilizes a value network as a learned heuristic to filter candidate rules, aiming to speed up convergence and improve the quality of the rules passed to the reasoning predictor. Finally, we have integrated attention-based interpretability with an interactive dashboard to provide mechanistic insights into model reasoning.

To validate the efficacy of these proposed methods, it is essential to subject them to empirical testing across diverse knowledge graphs. We hypothesize that: (1) **GRU** will excel in domains with smaller datasets due to its parameter-efficient sequence modeling, (2) **value-guided optimization** will yield more robust rule sets and faster convergence than standard EM by directly pruning the search space of low-quality logic, and (3) **attention weights** will reveal interpretable reasoning patterns correlated with rule performance. The following chapter details the experimental setup designed to test these hypotheses, describing the datasets selected, the evaluation metrics, and a comparative analysis of our models against the established baselines.

Chapter 5

Experiments and Evaluation

This chapter details the experimental evaluation of our proposed framework. The objective of these experiments is to empirically validate the architectural hypotheses introduced in the previous chapter. Specifically, we aim to measure:

1. **Generative Expressiveness:** Whether the GRU generator produces higher-quality rules than the standard LSTM.

2. **Optimization Efficacy:** Whether value-based reinforcement learning converges to better solutions than pure Expectation-Maximization (EM).

3. **Computational Efficiency:** The trade-off between training speed, memory footprint (measured via FLOPs), and predictive accuracy across different configurations.

5.1 Datasets

We selected three standard benchmark datasets to evaluate reasoning across different graph topologies and structural characteristics: **FB15k-237**, **UMLS**, and **Countries-S3**. These datasets are chosen to align with our project objectives: validating the efficiency of our proposed framework on large-scale general knowledge (FB15k-237) and smaller-scale (UMLS, Countries-S3) datasets.

- **FB15k-237:** A large-scale general knowledge graph derived from Freebase, covering diverse domains (movies, sports, geography, etc.). This subset removes inverse relations to prevent trivial rule learning. It tests the model's scalability and ability to generalize across heterogeneous relation types with moderate path lengths [23].

- **UMLS (Unified Medical Language System):** A biomedical knowledge graph representing relationships between medical concepts (e.g., *treats*, *diagnoses*). This dataset features hierarchical taxonomic structures (e.g., *isa* relations forming subsumption hierarchies) and tests compositional reasoning over medical ontologies [1].

- **Countries-S3:** A benchmark designed to test long-range multi-hop reasoning. The S3 variant removes easy direct paths, forcing the model to infer the *locatedIn* relation via long chains (e.g., *City* → *Region* → *Country* → *Continent*) [12].

Table 5.1 summarizes the statistics of these datasets, including structural properties relevant to our objectives.

Dataset	Entities	Relations	Train	Valid	Test
FB15k-237	14,541	237	272,115	17,535	20,466
UMLS	135	46	1,959	1,306	3,264
Countries-S3	271	2	985	24	24

Table 5.1: Basic statistics of the knowledge graph datasets [23, 1, 12].

These datasets span five orders of magnitude in training set size (985 to 272k triples). With our default rule length of 3-5 hops, we hypothesize that: (1) GRU's simpler architecture will excel on shorter reasoning chains and smaller datasets (UMLS, Countries-S3) due to reduced overfitting, (2) LSTM's greater capacity may provide advantages on larger datasets (FB15k-237), and (3) our value-guided optimization will consistently improve performance across all dataset scales.

5.2 Evaluation Metrics

We evaluate our contributions across two dimensions: predictive performance on link prediction tasks and computational efficiency of the training pipeline. These metrics enable comprehensive assessment of both effectiveness and efficiency of our architectural and optimization improvements.

5.2.1 Predictive Performance

- **MR (Mean Rank):** The average rank of the correct answer entity among all candidates. Lower values indicate better performance.
- **MRR (Mean Reciprocal Rank):** The average of the reciprocal ranks (1/rank) of the correct answer. This metric is less sensitive to outliers than MR. Higher values indicate better performance.
- **Hits@K (H@1, H@3, H@5, H@10):** The proportion of test queries for which the correct answer appears in the top K predicted entities. We report $K = 1, 3, 5, 10$. Higher values indicate better performance.
- **Δ MRR:** Relative improvement in MRR over the LSTM-1L+EM baseline, measuring the marginal gain from architectural or optimization changes.

5.2.2 Computational Efficiency

- **Process Time:** CPU time consumed by the training process, excluding I/O wait. Lower values indicate more efficient computation.
- **Total FLOPs:** Combined floating-point operations for complete generator + predictor forward pass, isolating the effect of architecture. Lower values indicate more efficient computation.

5.3 Results and Interpretation

We evaluate the original RNNLogic baseline (LSTM-1L+EM) against our three variants: **GRU+EM** (GRU generator with standard EM), **LSTM+RL** (LSTM with value-guided optimization), and **GRU+RL** (full combination). To assess scaling behavior, each configuration was tested with both 1-layer (1L) and 3-layer (3L) generators.

All experiments use identical hyperparameters and random seeds following the setup in Appendix D. Note that runtimes reflect varying hardware availability across experiments, limiting the comparability of the experiments. However, FLOPs provide a hardware-agnostic measure of computational complexity. Tables 5.2, 5.3 and 5.4 show complete results across datasets.

Model	Training	MRR	ΔMRR	MR	H@1	H@3	H@5	H@10	Proc.Time	Total FLOPs	Gen.Params	Pred.Params
LSTM-1L	EM	0.4306	—	2.54	0.000	0.875	0.958	1.000	24s	8.41M	1,578,755	5,488
LSTM-3L	EM	0.4444	+0.014	2.33	0.000	1.000	1.000	1.000	32s	14.71M	2,631,427	5,488
GRU-1L	EM	0.4458	+0.015	2.38	0.000	0.958	1.000	1.000	24s	6.44M	1,250,563	5,488
GRU-3L	EM	0.4479	+0.0173	2.33	0.000	0.958	1.000	1.000	29s	11.17M	2,040,067	5,488
LSTM-1L	RL	0.4380	-0.0074	2.88	0.083	0.792	0.958	1.000	27s	8.41M	1,578,755	5,488
LSTM-3L	RL	0.4066	-0.0240	2.79	0.000	0.792	0.958	1.000	35s	14.71M	2,631,427	5,488
GRU-1L	RL	0.5243	+0.0937	2.13	0.125	0.958	1.000	1.000	26s	6.44M	1,250,563	5,488
GRU-3L	**RL**	**0.7917**	**+0.3611**	**1.42**	**0.583**	**1.000**	**1.000**	**1.000**	31s	11.17M	2,040,067	5,488

Table 5.2: Results on Countries-S3. Best configurations per training method bolded. ΔMRR relative to LSTM-1L+EM baseline.

Model	Training	MRR	ΔMRR	MR	H@1	H@3	H@5	H@10	Proc.Time	Total FLOPs	Gen.Params	Pred.Params
LSTM-1L	EM	0.7602	—	6.47	0.671	0.831	0.873	0.904	9.6min	8.48M	1,612,591	706,088
LSTM-3L	EM	0.7545	-0.006	6.05	0.655	0.834	0.868	0.905	12.8min	14.78M	2,665,263	706,088
GRU-1L	EM	0.7394	-0.021	6.86	0.650	0.805	0.852	0.897	9.0min	6.51M	1,284,399	706,088
GRU-3L	EM	0.7543	-0.006	6.55	0.667	0.816	0.860	0.898	11.9min	11.24M	2,073,903	706,088
LSTM-1L	RL	0.7646	+0.004	6.32	0.678	0.832	0.863	0.897	10.7min	8.48M	1,612,591	706,088
LSTM-3L	RL	0.7707	+0.011	6.06	0.689	0.830	0.867	0.907	14.1min	14.78M	2,665,263	706,088
GRU-1L	RL	**0.7849**	**+0.025**	**5.40**	**0.712**	**0.833**	**0.875**	**0.913**	10.0min	6.51M	1,284,399	706,088
GRU-3L	RL	0.7558	-0.004	6.52	0.669	0.821	0.864	0.901	12.8min	11.24M	2,073,903	706,088

Table 5.3: Results on UMLS dataset. Best configuration bolded. ΔMRR relative to LSTM-1L+EM baseline.

Model	Training	MRR	ΔMRR	MR	H@1	H@3	H@5	H@10	Proc.Time	Total FLOPs	Gen.Params	Pred.Params
LSTM-1L	EM	0.3221	—	376.00	0.2377	0.3521	0.4119	0.4921	1h05m	9.15M	1,941,723	792,782
LSTM-3L	EM	0.3207	-0.0014	379.26	0.2344	0.3524	0.4132	0.4931	1h28m	15.45M	2,994,395	792,782
GRU-1L	EM	0.3236	+0.0015	363.24	0.2392	0.3546	0.4132	0.4914	1h02m	7.18M	1,613,531	792,782
GRU-3L	EM	0.3168	-0.0053	491.46	0.2328	0.3487	0.4059	0.4846	1h21m	11.25M	2,403,035	792,782
LSTM-1L	**RL**	**0.3288**	**+0.0067**	**343.97**	**0.2458**	**0.3604**	**0.4166**	**0.4939**	1h14m	9.15M	1,941,723	792,782
LSTM-3L	RL	0.3186	-0.0035	361.68	0.2343	0.3493	0.4065	0.4856	1h37m	15.45M	2,994,395	792,782
GRU-1L	RL	0.3263	+0.0042	347.33	0.2425	0.3585	0.4150	0.4935	1h09m	7.18M	1,613,531	792,782
GRU-3L	RL	0.3207	-0.0014	381.96	0.2362	0.3525	0.4100	0.4876	1h28m	11.25M	2,403,035	792,782

Table 5.4: Results on FB15k-237 dataset. Best configuration bolded. ΔMRR relative to LSTM-1L+EM baseline.

Table 5.5 aggregates the best configuration per dataset, highlighting the consistent superiority of RL-optimized models across all three benchmarks.

Dataset	Best Config	MRR	Δ vs. Baseline
Countries-S3 (Small, Simple)	GRU-3L+RL	0.7917	+0.3611
UMLS (Middle Ground)	GRU-1L+RL	0.7849	+0.0247
FB15k-237 (Large, Complex)	LSTM-1L+RL	0.3288	+0.0067

Table 5.5: Best-performing configurations per dataset.

5.3.1 Interpretation of Results

Our results reveal consistent patterns across datasets of varying scale and complexity: (1) RL optimization universally outperforms EM, and (2) GRU architectures enable better performance for smaller datasets than LSTM. The following analysis explains these patterns and their implications for rule learning.

1. **RL universally improves over EM baseline.** RL-optimized configurations dominate all datasets: +84% MRR on Countries-S3, +3.3% on UMLS, +2.1% on FB15k-237. These diminishing returns suggest either that RL excels most on smaller, logically-structured graphs or RNNLogic nears performance saturation on the other datasets, limiting further gains. Either way, these results validate the performance improvement due to RL's more efficient rule selection mechanism.

2. **Optimal generator depends on dataset scale.** While RL universally improves performance, generator choice depends critically on dataset size. GRU dominates smaller datasets (Countries-S3: GRU-3L+RL; UMLS: GRU-1L+RL) while LSTM excels on large-scale FB15k-237 (LSTM-1L+RL: 0.3288 MRR). This suggests lightweight models like GRU suit smaller datasets and LSTM scales better to larger graphs. This warrants future tests with larger architectures like Mamba on larger datasets.

3. **GRU consistently more parameter-efficient.** GRU variants consistently achieve top performance with 20-25% fewer parameters and FLOPs than LSTM equivalents. On UMLS, **GRU-1L+RL** delivers state-of-the-art results with 6.51M FLOPs vs 8.48M FLOPS at baseline. Similarly, **GRU-3L+RL** delivers the best results for the Countries-S3 dataset with 6.44M FLOPS vs 14.71M FLOPS for the LSTM-3L configuration.

4. **RL processing times higher despite superior performance.** While RL configurations achieve the best MRR for all datasets, they exhibit longer processing times than EM baselines. This occurs because we kept the value network output size identical to EM (same number of candidate rules), even though RL's predictive value estimation should theoretically select fewer, higher-quality rules due to its ability to prioritize promising candidates early. Having different hardware setups across experiments also make direct processing time comparisons unreliable. FLOPs measurements provide the only reliable, hardware-independent measure of computational complexity. Future work optimizing K (top-K selection) and running on identical machines would enable fairer timing comparisons and likely reduce RL overhead.

5.3.2 Attention and Grounding Analysis

We apply the grounding-based link contribution analysis (Section 4.5.2) to the rules generated by our best-performing configuration on UMLS (GRU-1L+RL, MRR 0.7849). The analysis covers 44 query relations and evaluates the top 10 rules per relation (by H-score), producing 440 rule–relation pairs with step-by-step grounding metrics on the UMLS test set.

Position importance follows a late-hop pattern. Table 5.6 reports the average precision, coverage, and contribution at each position across all analysed rules. Position 0 (the head relation) and position 1 (the first body hop) consistently show zero precision: at these early stages, the entity set has not yet been filtered enough to isolate the correct answer. Meaningful precision first appears at position 2 (average 0.117) and increases through positions 3 and 4 (0.214 and 0.393 respectively). This is consistent with the intuition that early hops serve as structural anchors that establish reachability, while later hops carry the actual discriminative signal that narrows the candidate set toward the correct answer. Coverage, conversely, decreases from 1.0 at position 0 to around 0.28–0.31 at later positions, reflecting the progressive pruning of paths that dead-end on the graph.

Position	Avg Precision	Avg Coverage	Avg Contribution
0 (Head)	0.000	1.000	0.000
1 (Body 1)	0.000	0.840	0.000
2 (Body 2)	0.117	0.294	0.101
3 (Body 3)	0.214	0.280	0.164
4 (Body 4)	0.393	0.312	0.206

Table 5.6: Average grounding metrics by position in the rule chain, aggregated over all 440 analysed rules on UMLS. Later positions exhibit higher precision, indicating that the discriminative signal concentrates in the final hops.

Certain relations are structurally more useful for reasoning. When aggregating contribution by relation type (across all positions and rules), several UMLS relations stand out. *process_of* has the highest average contribution (0.292), followed by *affects* (0.196), *result_of* (0.189), and *manifestation_of* (0.170). These relations tend to be both highly connected in the UMLS graph and semantically informative for biomedical reasoning chains. Relations such as *part_of* and *degree_of* contribute less on average, likely because they connect fewer entity pairs or lead to dead-end subgraphs. This per-relation breakdown, visualised in the Positions and Relations tab of the dashboard, can inform future work on relation-aware rule generation or pruning strategies.

Performance varies across query relations. Of the 44 query relations evaluated, 15 achieve perfect precision (1.0) for their top-ranked rule, including high-frequency relations such as *result_of* (282 test queries), *process_of* (223), *produces* (154), and *manifestation_of* (86). At the other end, 5 relations — *adjacent_to*, *assesses_effect_of*, *carries_out*, *conceptual_part_of*, and *connected_to* — have zero precision across all top-10

rules, meaning no generated rule successfully grounds a correct answer for any test query of that type. These zero-performance relations tend to have very few test triples (2–33), sparse graph connectivity, or relation types that are difficult to reach through chain-like reasoning paths. This finding highlights a practical limitation of path-based rule learning: certain relation types may require non-chain rule templates (e.g., rules with branching or constraints) to be captured effectively.

H-score alignment with actual precision. The dashboard's Diagnostics tab reveals the relationship between the model's H-score (its internal confidence in a rule) and the rule's actual grounding precision. The alignment is imperfect: some rules with strongly negative H-scores (indicating the generator considers them unlikely) achieve perfect precision, while some rules with relatively high H-scores have zero actual precision. This is partially expected, since the H-score reflects the posterior probability under the generator and predictor jointly, which conflates rule plausibility (how likely the generator is to produce it) with rule utility (how well it performs on test data). The grounding analysis provides the ground-truth signal that the H-score alone cannot: it measures what the rules actually do on the graph, independent of the model's opinion.

Knowledge graph visualisation as a validation tool. The Rule Explorer tab of the dashboard enables concrete validation of individual rules. For a selected rule, it performs a depth-first search on the uploaded UMLS triples to find a concrete grounding path. Figure 4.3 shows an example for the rule *prevents* → *prevents* → *complicates* → *result_of* → *manifestation_of*: the dashboard finds a path through actual biomedical entities (e.g., *therapeutic_or_preventive_procedure* → *disease_or_syndrome* → *experimental_model_of_disease* → *acquired_abnormality* → *pathologic_function*), and additionally shows that a direct *prevents* edge exists between the start and end entities. This makes it possible to verify, at the level of individual entities, that a rule captures a medically meaningful reasoning chain rather than a statistical artifact.

5.4 Summary

Our experiments across three datasets demonstrate that **RL optimization universally improves over EM baselines** (+84% Countries-S3, +3.3% UMLS, +2.1% FB15k-237) while **GRU architectures consistently outperform LSTM in terms of efficiency with 20–25% fewer parameters/FLOPs**. Optimal configurations vary by dataset size: GRU+RL dominates smaller datasets, while LSTM+RL leads large-scale FB15k-237. Additionally, we used unoptimized hyperparameters and kept value network output size identical to EM baselines. Tuning K and shrinking rule output could substantially reduce compute costs while maintaining accuracy gains. Attention analysis also reveals that attention weights better capture interpretable patterns across different relations within logical chains. The following chapter discusses these findings' implications and future research directions. The next chapter explores these findings' broader implications and future research directions.

Chapter 6

Conclusion and Perspectives

This report has presented a comprehensive re-engineering of the RNNLogic framework for knowledge graph completion, targeting three specific limitations of the original system: the parameter overhead of the LSTM-based rule generator, the computational cost of exhaustive rule evaluation in the EM loop, and the lack of fine-grained interpretability beyond whole-rule H-scores. We addressed these through a GRU-based generator, a value-guided EM optimization strategy, and an attention-driven grounding analysis pipeline. Our experimental evaluation across three benchmark datasets (Countries-S3, UMLS, FB15k-237) allows us to disentangle the contributions of each modification and identify their relative impact.

6.1 Disentangling Architectural and Optimization Contributions

A central question raised by our experimental design is whether the observed performance gains originate primarily from the change in generator architecture (GRU vs. LSTM) or from the switch in optimization strategy (value-guided RL vs. standard EM). Tables 5.2–5.4 provide the evidence to answer this question by evaluating all four combinations: LSTM+EM (baseline), GRU+EM, LSTM+RL, and GRU+RL.

Impact of the GRU architecture alone. Comparing GRU+EM against the LSTM+EM baseline isolates the effect of the architectural change. On Countries-S3, GRU-1L+EM achieves an MRR of 0.4458 versus 0.4306 for LSTM-1L+EM (ΔMRR = +0.015). On FB15k-237, GRU-1L+EM yields 0.3236 versus 0.3221 (ΔMRR = +0.0015). On UMLS, however, GRU-1L+EM slightly underperforms at 0.7394 versus 0.7602 (ΔMRR = -0.021). In all cases, the GRU achieves these results with 20–25% fewer parameters and FLOPs than the corresponding LSTM configuration. The architectural change therefore delivers consistent efficiency gains but only marginal and inconsistent improvements in predictive accuracy when used with standard EM.

Impact of value-guided RL alone. Comparing LSTM+RL against LSTM+EM isolates the effect of the optimization change. The RL-optimized LSTM improves on all three datasets: +0.0074 MRR on Countries-S3 (LSTM-1L), +0.0044 on UMLS (LSTM-1L), and +0.0067 on FB15k-237 (LSTM-1L). On UMLS, the deeper LSTM-3L+RL configuration reaches 0.7707, a +0.011 improvement over the LSTM-3L+EM baseline. Unlike the architectural change, the RL optimization provides consistent gains across all datasets and scales, confirming that the value network's ability to pre-filter candidate rules before expensive predictor evaluation leads to higher-quality rule sets.

The dominant contribution is optimization, amplified by architecture. Across all three datasets, the value-guided RL optimization is the single most impactful modification. It produces consistent improvements regardless of the underlying generator architecture. The GRU architecture, while valuable for efficiency, only unlocks its full predictive potential when combined with RL. This synergy is most visible on Countries-S3, where GRU-3L+RL achieves 0.7917 MRR (an 84% relative improvement over the LSTM-1L+EM baseline) while GRU-3L+EM achieves only 0.4479. The pattern is clear: RL provides the quality signal, and GRU provides the efficient substrate on which that signal acts most effectively for smaller datasets.

6.2 Best Final Configurations

Table 5.5 summarises the best-performing configuration for each dataset:

- **Countries-S3** (small, simple graph): **GRU-3L+RL** with MRR 0.7917 (ΔMRR = +0.3611 over baseline). The combination of a deeper GRU generator and value-guided optimization proves decisive for long-range multi-hop reasoning on small graphs.
- **UMLS** (medium, biomedical graph): **GRU-1L+RL** with MRR 0.7849 (ΔMRR = +0.0247). A single-layer GRU suffices for the short reasoning chains in the biomedical domain, and RL provides meaningful gains over EM.
- **FB15k-237** (large, heterogeneous graph): **LSTM-1L+RL** with MRR 0.3288 (ΔMRR = +0.0067). On the largest and most complex dataset, LSTM's greater representational capacity outperforms GRU, though the improvement from RL remains the dominant factor.

A consistent pattern emerges: GRU+RL dominates on smaller datasets where parameter efficiency prevents overfitting, while LSTM+RL leads on larger datasets where the additional capacity of the LSTM is beneficial. In all cases, RL optimization is present in the best configuration, reinforcing its role as the primary driver of performance gains.

6.3 Attention and Grounding Analysis

Beyond predictive performance, our grounding-based link contribution analysis (Section 4.5.2) applied to the best UMLS configuration (GRU-1L+RL) reveals structural insights into how rules operate on the knowledge graph. Three key findings emerged from the analysis of 440 rule–relation pairs:

1. **Late hops carry the discriminative signal.** Average precision increases from 0.0 at positions 0–1 to 0.117 at position 2 and 0.393 at position 4, while coverage decreases from 1.0 to approximately 0.31. Early hops serve as structural anchors establishing reachability; later hops perform the actual filtering that isolates the correct answer.

2. **Certain relations are structurally more useful.** Relations such as `process_of` (average contribution 0.292), `affects` (0.196), and `result_of` (0.189) consistently contribute more to correct predictions, suggesting that relation-aware rule generation or pruning could improve future systems.

3. **H-score calibration is imperfect.** The diagnostics tab of our dashboard reveals cases where rules with low H-scores achieve perfect grounding precision, and vice versa. This confirms the value of our dual-view approach: the model's internal confidence (H-score) and the empirical ground truth (grounding precision) provide complementary signals, and relying on either alone is insufficient.

The interactive Streamlit dashboard developed as part of this contribution makes these analyses accessible for both research validation and live demonstration, integrating knowledge graph subgraph visualisation with per-link metric annotations.

6.4 Limitations

Despite these advances, our work faces several limitations:

- **Unoptimised hyperparameters:** We used identical hyperparameters across all configurations (N=4000, K=1500) without dataset-specific tuning. The value network output size was kept identical to the EM baseline, meaning the theoretical speedup from pre-filtering was not fully exploited. Tuning K and shrinking the rule output could reduce computational cost while maintaining accuracy gains.

- **Hardware variability:** Processing times were measured across different hardware setups, limiting direct timing comparisons. FLOPs provide the only reliable hardware-independent measure of computational complexity in our experiments.

- **RL stability:** The value-guided optimization introduces additional hyperparameters (value network learning rate, selection count K) that require careful tuning. While effective, the RL variant is more sensitive to these choices than the stable EM algorithm.

- **Grounding analysis scope:** Our link contribution analysis was applied only to UMLS. Extending it to FB15k-237 would test whether the observed late-hop pattern and relation-level findings generalise to larger, more heterogeneous graphs.

- **Zero-precision relations:** Five UMLS relations yielded zero precision across all top-10 rules, indicating that chain-shaped rule templates cannot capture all relation types. Non-chain templates with branching or constraints may be required for these cases.

6.5 Future Work

Building on these findings, we identify three directions for future research, organised around our three contributions:

6.5.1 Architecture Scaling and Refinement

Our results show that GRU excels on small-to-medium datasets while LSTM leads on larger graphs. A natural next step is to evaluate intermediate architectures or adaptive mechanisms that select generator depth and type based on dataset characteristics. Additionally, the exploratory Mamba and HTRM architectures (Appendix B) showed promise on specific benchmarks but were not combined with RL optimisation due to computational constraints; doing so could reveal further gains.

6.5.2 RL Optimisation Refinement

The value network currently uses a fixed architecture (bidirectional GRU with MLP head). Exploring alternative value estimators, curriculum-based K scheduling (starting with large K and progressively reducing it), and adaptive N/K ratios could improve both convergence speed and final rule quality. The variational inference approach explored in Appendix B.5 also showed initial promise and could be revisited as a complementary optimisation strategy.

6.5.3 Quantitative Explainability Metrics

Our current explainability pipeline provides qualitative insights through the dashboard. Developing quantitative metrics—such as attention–grounding alignment scores, rule fidelity measures, and per-relation predictability indices—would enable systematic comparison of explainability across models and datasets. These metrics could also serve as auxiliary training objectives, guiding the generator toward rules that are not only predictive but also interpretable.

6.6 Project Contributions

This project has delivered three integrated contributions to the RNNLogic framework: (1) a GRU-based rule generator that achieves 20–25% parameter reduction while matching or exceeding LSTM performance on smaller datasets, (2) a value-guided EM optimisation strategy that consistently improves predictive accuracy across all tested datasets and constitutes the single most impactful modification, and (3) an attention-based grounding analysis pipeline with an interactive dashboard that provides both model-level and data-level explainability for learned rules. Together, these contributions advance the RNNLogic framework toward more efficient, more accurate, and more interpretable neurosymbolic reasoning on knowledge graphs.

Appendix A

Learned Rules

To demonstrate the interpretability of our framework, we present examples of logic rules generated on the FB15k-237, UMLS, and Countries S3 datasets. Unlike black-box embedding models, these symbolic outputs allow humans to verify the reasoning logic.

A.1 FB15k-237 Dataset

Table A.1 shows representative rules learned from the Freebase knowledge graph. These capture compositional patterns typical of large-scale KGs.

Target Relation	**Learned Rule Body**
location/location	$\leftarrow$ *organization/headquarters*(X, Z) $\wedge$ *location/location*(Z, Y)
profession	$\leftarrow$ *people/person/profession*(X, Z) $\wedge$ *profession*(Z, Y)
place/containedby	$\leftarrow$ *place/location/contains*(Y, Z) $\wedge$ *place/containedby*(Z, X)
employer	$\leftarrow$ *employment/employer*(X, Z) $\wedge$ *employer*(Z, Y)

Table A.1: Compositional rules generated on the FB15k-237 dataset.

A.2 UMLS Dataset

Table A.2 shows biomedical rules derived from the Unified Medical Language System. The model effectively captures causal and hierarchical dependencies found in the training data.

Target Relation	**Learned Rule Body**
diagnoses	$\leftarrow \text{treats}(X, Y) \wedge \text{prevents}(X, Y)$
isa	$\leftarrow \text{part\_of}(X, Z) \wedge \text{part\_of}(Z, Y)$
affects	$\leftarrow \text{causes}(X, Z) \wedge \text{affects}(Z, Y)$
interacts_with	$\leftarrow \text{isa}(X, Z) \wedge \text{interacts\_with}(Z, Y)$
location_of	$\leftarrow \text{part\_of}(X, Z) \wedge \text{location\_of}(Z, Y)$

Table A.2: Hierarchical and causal rules generated on the UMLS dataset.

A.3 Countries S3 Dataset

Table A.3 demonstrates rules learned for geopolitical reasoning. The model successfully captures the transitivity of location and the symmetry of borders. Table A.3 demonstrates rules learned for geopolitical reasoning. The model successfully captures the transitivity of location and the symmetry of borders.

Target Relation	**Learned Rule Body**
locatedin	$\leftarrow \text{locatedin}(X, Z) \wedge \text{locatedin}(Z, Y)$
locatedin	$\leftarrow \text{neighbor}(X, Z) \wedge \text{locatedin}(Z, Y)$
neighbor	$\leftarrow \text{neighbor}(Y, X)$
neighbor	$\leftarrow \text{locatedin}(X, Z) \wedge \text{neighbor}(Z, Y)$

Table A.3: Transitive and symmetric rules generated on the Countries S3 dataset.

Appendix B

Exploratory Approaches

During the development of our core pipeline, we investigated several alternative strategies. Although excluded from the final work, these explorations yielded important insights about data characteristics, system bottlenecks, and the robustness of our primary approach.

B.1 Semantic Rule Miner

We experimented with a Semantic Rule Miner to enrich the initial rule pool before EM training via a hybrid approach:

1. **Statistical Sampling:** Analyzing triplet distributions to identify evidence of symmetry, transitivity, or inverse relations.

2. **LLM Integration:** Using a Large Language Model (e.g., Ollama) to infer properties from relation labels (e.g., *parent_of* as inverse of *child_of*).

The outputs were reconciled using heuristic confidence rules and added to the initial pool. However, semantic rules were extracted in very low numbers, which was orders of magnitude fewer than path-mined rules, causing them to be drowned out. Even with clustering or top-N selection to limit original rules, semantic rules yielded no performance gains. We thus discontinued further exploration.

B.2 Biased Random Walk Miner

We explored replacing the exhaustive DFS miner with a biased random walk [16], which balances breadth-first and depth-first exploration to scale to larger graphs. Initial implementation showed comparable speed to exhaustive search on medium-scale KGs. However, we prioritized enhancements to the core RNNLogic flow over further miner development.

B.3 Rule Clustering and Top-N Rule Selection

To reduce the rule search space, we explored two strategies: clustering similar rules and retaining only the top-N rules based on confidence scores. Both methods aimed to simplify the rule pool by eliminating redundancy and focusing on the most representative patterns. However, both approaches ultimately degraded performance.

In the clustering approach, rules generated by the path-based miner were grouped so that only representative centroids from each cluster were retained. This was expected to preserve semantic diversity while preventing overlapping rules from overshadowing more

meaningful ones, including the newly introduced semantic rules we tested out. Yet, performance declined when the model reasoned with these clustered representatives. The loss of subtle variations across similar rules appeared to be detrimental, suggesting that the model benefited from having multiple slight variations of the same rule type.

Similarly, the Top-N selection method filtered rules strictly according to their confidence scores. Although intuitive, this pruning removed many moderately scoring but complementary rules that contributed essential variation during reasoning. As a result, this approach also reduced overall performance..

These findings highlight that having a larger, more redundant rule pool is advantageous. The observed performance gain arises not from individual rule precision but from the diversity in rules, which enrich the model's reasoning capacity.

B.4 Variational Inference (VI) vs. Expectation-Maximization (EM)

We refactored the original optimization loop from Expectation-Maximization (EM) to Variational Inference (VI), treating logic rules as latent variables and maximizing the Evidence Lower Bound (ELBO) to accelerate convergence [14]. Preliminary experiments on a sampled FB15k-237 dataset showed promising performance, with VI achieving slightly better performance. Theoretically, VI offers scalability advantages over EM on larger datasets. However, we prioritized other experimental approaches. Future work could validate VI's efficacy on full-scale knowledge graphs.

B.5 Additional Generator Architectures

In addition to our core GRU and RL approach, we evaluated several alternative generator architectures during development. These explorations provided insights into architectural scaling but were excluded from the main results due to either compute limitations or unpromising preliminary results. Table B.1 shows the preliminary performance of different model configurations across datasets.

We evaluated several alternative generator architectures during development. Specifically, we observed that:

- **Mamba**, a state-space model known for its ability to capture long-range dependencies with linear-time complexity, was selected as a modern alternative to recurrent architectures. It consistently improves over the LSTM baseline in terms of MRR, but its training time is 2–4× slower than GRU+RL on all datasets. In practice, this makes it less attractive when both accuracy and efficiency matter.

- **HTRM** (Hierarchical Transformer), chosen for its capacity to model multi-hop relational patterns through hierarchical attention over rule structures, achieves the highest MRR on UMLS (0.7875). However, this gain comes with a substantial compute cost (28m vs. 9m for GRU-RL-1L), suggesting that HTRM is a promising but currently too heavy option for our setting.

Dataset	Model	MR	MRR	Hits@1	Hits@3	Hits@10	Speed
Kinship	Original	4.306195	0.5893	0.3987	0.6786	0.9158	12m
	GRU-3L	3.5574	0.6311	0.4722	0.7410	0.9407	9m
	GRU-RL-1L	3.5680	**0.6319**	0.4733	0.7410	0.9407	8m
	GRU-RL-3L	3.5843	**0.6337**	0.4775	0.7400	0.9407	11m
	Mamba	3.5152	**0.6343**	0.4851	0.7299	0.9442	16m
	LSTM-RL-1L	3.4426	0.6192	0.4488	0.7385	0.9513	10m
	LSTM-RL-3L	3.4852	0.6179	0.4475	0.7351	0.9495	12m
	HTRM	3.5601	0.6312	0.4810	0.7245	0.9405	15m
	T-XL	3.5546	0.6324	0.4825	0.7256	0.9416	14m
	GPT2	4.1280	0.5509	0.3642	0.6678	0.9324	6m
UMLS	Original	6.5376	0.7473	0.6455	0.8275	0.9017	13m
	GRU-3L	7.3254	0.7649	0.6847	0.8235	0.8989	11m
	GRU-RL-1L	6.4424	**0.7805**	0.7004	0.8438	0.9118	9m
	GRU-RL-3L	6.8398	0.7696	0.6761	0.8186	0.9112	15m
	Mamba	6.6780	**0.7747**	0.6903	0.8428	0.9090	32m
	LSTM-RL-1L	7.2850	0.7668	0.6881	0.8255	0.9001	11m
	LSTM-RL-3L	7.4197	0.7652	0.6814	0.8287	0.9025	11m
	HTRM	6.4520	**0.7875**	0.7105	0.8510	0.9150	28m
	T-XL	7.1664	0.7613	0.6801	0.8195	0.9022	23m
	GPT2	8.5200	0.6917	0.5850	0.7620	0.8850	10m
Countries-S3	Original	2.5416	0.4305	0.0000	0.8750	1.0000	37s
	GRU-3L	3.0833	0.4549	0.1666	0.7083	1.0000	26s
	GRU-RL-1L	2.1250	**0.5243**	0.1250	0.9583	1.0000	28s
	GRU-RL-3L	1.4166	**0.7917**	0.5833	1.0000	1.0000	32s
	Mamba	2.5833	**0.4983**	0.1667	0.7917	1.0000	4m
	LSTM-RL-1L	3.7917	0.4678	0.0000	0.5833	0.9583	40s
	LSTM-RL-3L	3.3750	0.4128	0.0417	0.7083	1.0000	48s
	HTRM	2.7083	0.4136	0.0000	0.8750	1.0000	4m
	T-XL	2.8333	0.4400	0.0833	0.7917	1.0000	25s
	GPT2	2.7083	0.4410	0.0000	0.9167	1.0000	28s

Table B.1: Experimental Results. Best MRR results for each dataset are bolded.

- **TransformerXL**, selected for its segment-level recurrence mechanism that enables modeling longer rule sequences without truncation, is competitive on Kinship and UMLS. However, it never clearly surpasses GRU+RL in either accuracy or runtime, so it does not justify the added architectural complexity.

Taken together, these results indicate that alternative generators either (i) offer only marginal accuracy gains at much higher cost or (ii) fail to improve over GRU+RL. This empirically supports our choice of GRU+RL as the primary architecture, since it provides the best balance between predictive performance and computational efficiency across datasets.

Appendix C

Semantic Rule Mining Analysis

In this section, we provide more details about our work on the **Semantic Rule Miner** module, which is designed to augment the model's neural reasoning capabilities by extracting symbolic patterns,specifically transitive, inverse, and symmetric rules, from knowledge graphs. We further analyze the impact of rule set size and redundancy on performance, specifically reporting on failed attempts to optimize the rule pool through clustering and confidence-based filtering. These experiments highlight the critical importance of rule diversity.

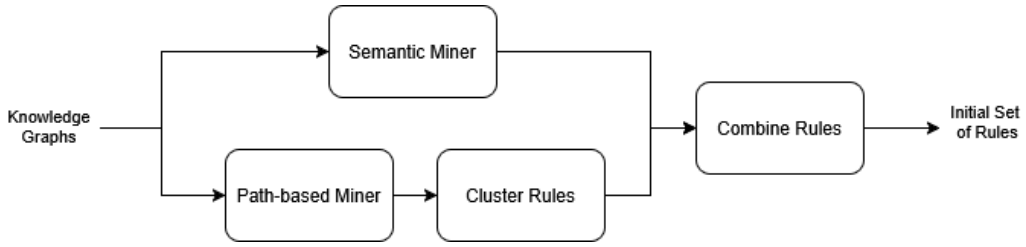

Figure C.1: **Overview of the Updated Rule Miner.** The semantic miner extracts symbolic patterns (transitive, symmetric, inverse) from the knowledge graph and combines them with the pruned rules from the path-based miner to guide reasoning.

C.1 Pruning Approaches

Since the number of mined semantic rules was far smaller than the original path-based set, we explored two pruning methods to prevent the semantic rules from being overshadowed: (1) clustering similar rules and (2) applying a Top-N filter based on confidence. Both methods aimed to simplify the rule pool by eliminating redundancy and focusing on the most representative patterns. However, as detailed below, both approaches ultimately degraded performance, leading to a key insight regarding the value of redundancy.

C.1.1 Clustering Approach

In the clustering strategy, rules generated by the path-based miner were grouped semantically, and only representative centroids from each cluster were retained. The hypothesis was that this would preserve semantic diversity while preventing overlapping rules from overshadowing distinct patterns. However, performance declined significantly when the model reasoned with these clustered representatives. The loss of subtle variations across similar rules appeared to be detrimental. Our analysis suggests that the model benefits from having multiple slight variations of the same rule type, as these redundant paths provide robust evidence during the reasoning process.

C.1.2 Top-N Confidence Selection

Similarly, the Top-N selection method filtered rules strictly according to their confidence scores, retaining only the top 10% of rules. Although intuitive, this pruning removed many moderately scoring but complementary rules that contributed essential variation. As a result, this approach also reduced overall performance, failing to capture the long tail of reasoning patterns required for difficult queries.

C.2 Results

Table C.1 presents the quantitative impact of these pruning strategies on the FB15k-237 and Kinship datasets. The original baseline represents the model using the full rule set.

Dataset	**Full Rule Set (Baseline)**	**Clustered Rules (10%)**		**Top-N Rules (10%)**		**Full Set**
		Raw	**+ Semantic**	**Raw**	**+ Semantic**	**+ Semantic**
FB15k-237	0.3179	-	0.1421	-	0.2256	0.3135
Kinship	0.5893	0.5239	0.5239	0.5791	0.5791	0.5893

Table C.1: Performance comparison (MRR) of rule pruning strategies. The clustered approach causes a severe drop in performance (e.g., 0.3179 → 0.1421 on FB15k-237), confirming that removing redundant variations hurts reasoning. **Top-N** pruning is less destructive but still fails to match the full set on Kinship.

These findings highlight that having a larger, more redundant rule pool is advantageous. The observed performance gain arises not from individual rule precision but from the diversity in rules, which enrich the model's reasoning capacity. Ultimately, the performance drop observed in these configurations, led us to scrap this experiment entirely.

Appendix D

Experimental Configuration

This appendix provides complete experimental configuration details to ensure full reproducibility of our results. We document all key hyperparameters (Table D.1). These settings enable exact replication and support future hyperparameter optimization or extension to new knowledge graphs.

Parameter	Value
General Settings	
Save path:	`<save_directory>`
Load path:	`<optional>`
Data path:	`../data/<dataset>`
Rule file:	`../data/<dataset>/mined_rules.txt`
Data batch size:	32
Random seed:	1
GPU:	0
Expectation–Maximization (EM) Configuration	
EM iterations:	1
Prior weight (λ):	0.001
Number of rules:	100
Maximum rule length ($L_{\max}$):	3
Generator Network	
Embedding dimension:	512
Hidden dimension:	256
Number of layers:	1
Optimizer:	Adam
Batch size:	512
Pre-train epochs:	10 000
Pre-train learning rate:	1×10^{-3}
Pre-train print frequency:	Every 1 000 steps
Train epochs:	100
Train learning rate:	1×10^{-5}
Train print frequency:	Every 1 000 steps
Post-train epochs:	1 000
Post-train learning rate:	1×10^{-5}
Post-train print frequency:	Every 1 000 steps
Predictor Network	
GPU(s):	[0]
Entity feature type:	Bias
Optimizer:	Adam
Learning rate:	1×10^{-3}
Weight decay:	0
Label smoothing:	0.2
Batches per epoch:	1 000 000
Print frequency:	Every 1 000 steps
Evaluation expectation:	True
H-score print frequency:	Every 1 000 steps
Final Prediction Phase	
Number of iterations:	5
Number of rules:	[100]
Maximum rule length: ($L_{\max}$)	[3]
PredictorPlus Network	
GPU(s):	[0]
Hidden dimension:	16
Optimizer:	Adam
Learning rate:	5×10^{-3}
Weight decay:	0
Label smoothing:	0.2
Batches per epoch:	1 000 000
Print frequency:	Every 1 000 steps
Evaluation expectation:	True

Table D.1: Full experimental configuration with general, EM, generator, predictor, and predictor-plus hyperparameters.

Bibliography

[1] Olivier Bodenreider. The unified medical language system (umls): integrating biomedical terminology. Nucleic acids research, 32(suppl$_1$) : $D267 - -D270$, 2004. (*Citedonpages* 31*and* 32.)

[2] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In Neural Information Processing Systems, 2013. (Cited on page 4.)

[3] Kewei Cheng, Nesreen K. Ahmed, and Yizhou Sun. Neural compositional rule learning for knowledge graph reasoning, 2023. (Cited on page 8.)

[4] Kewei Cheng, Jiahao Liu, Wei Wang, and Yizhou Sun. Rlogic: Recursive logical rule learning from knowledge graphs. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD '22, page 179–189, New York, NY, USA, 2022. Association for Computing Machinery. (Cited on page 8.)

[5] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches, 2014. (Cited on page 22.)

[6] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014. (Cited on page 22.)

[7] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets, 2020. (Cited on page 19.)

[8] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In Proceedings of the 6th International Conference on Learning Representations (ICLR). ICLR, 2018. (Cited on page 7.)

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural Computation, 9(8):1735–1780, 1997. (Cited on page 16.)

[10] Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. A survey on knowledge graphs: Representation, acquisition, and applications, February 2022. (Cited on page 1.)

[11] Ni Lao and William W. Cohen. Random walk inference and learning in a large scale knowledge base. Machine Learning, 81(1):53–67, 2010. (Cited on page 6.)

[12] Zhenfeng Lei. Kgdatasets: Experimental data sets over knowledge graph. https://github.com/ZhenfengLei/KGDatasets, 2019. (Cited on pages 31 and 32.)

[13] Xi Victoria Lin, Richard Socher, and Caiming Xiong. Multi-hop knowledge graph reasoning with reward shaping, 2018. (Cited on page 7.)

[14] Yankai Liu, Zhiyuan Wang, Jiaxuan Zhao, and Maosong Liu. Variational knowledge graph reasoning. In NAACL-HLT, 2018. (Cited on page 44.)

[15] 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, 2024. (Cited on page 8.)

[16] Duong Nguyen and Fragkiskos D. Malliaros. Biasedwalk: Biased sampling for representation learning on graphs, 2018. (Cited on page 43.)

[17] Meng Qu, Junkun Chen, Louis-Pascal Xhonneux, Yoshua Bengio, and Jian Tang. Rnnlogic: Learning logic rules for reasoning on knowledge graphs, 2021. (Cited on pages 2, 4, 5, 9, 11, 15 and 21.)

[18] Matthew Richardson and Pedro Domingos. Markov logic networks. Machine Learning, 62(1–2):107–136, 2006. (Cited on page 6.)

[19] Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. Drum: End-to-end differentiable rule mining on knowledge graphs, 2019. (Cited on page 6.)

[20] Yelong Shen, Jianshu Chen, Po-Sen Huang, Yuqing Guo, and Jianfeng Gao. M-walk: Learning to walk over graphs using monte carlo tree search, 2018. (Cited on page 7.)

[21] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. Nature, 529(7587):484–489, 2016. (Cited on page 24.)

[22] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space, 2019. (Cited on pages 5 and 19.)

[23] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality, pages 57–66, 2015. (Cited on pages 31 and 32.)

[24] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction, 2016. (Cited on page 5.)

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Advances in Neural Information Processing Systems, volume 30, 2017. (Cited on page 26.)

[26] Bishan Yang, Wen tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases, 2015. (Cited on page 5.)

[27] Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base reasoning, 2017. (Cited on page 6.)