Beyond Factual Knowledge: Benchmarking and Learning Step-Level Procedural Rule Reasoning in Large Language Models

Bohan Yu[1,2], **Pengfei Cao**[2,3*], **Chen Han**[1,6], **Chenxi Zhou**[1,2], **Zhiheng Zhang**[2,3], **Zhiyang Xie**[2,3], **Wenhao Teng**[4], **Xiangwen Liao**[5], **Jun Zhao**[2,3], **Kang Liu**[2,3*]
[1]School of Advanced Interdisciplinary Sciences, University of Chinese Academy of Sciences
[2]The Key Laboratory of Cognition and Decision Intelligence for Complex Systems, Institute of Automation, Chinese Academy of Sciences
[3]School of Artificial Intelligence, University of Chinese Academy of Sciences
[4]Department of Gastrointestinal Surgery, Fujian Provincial Cancer Hospital
[5]College of Computer and Data Science, Fuzhou University
[6]Academy of Mathematics and Systems Science, Chinese Academy of Sciences
yubohan2025@ia.ac.cn {pengfei.cao,jzhao,kliu}@nlpr.ia.ac.cn

Abstract

Large language models (LLMs) excel at text understanding and generation, yet still struggle to reliably understand and apply externally provided procedural rules at scale. To evaluate this capability, we introduce **RuleWorld**, a large-scale benchmark that reformulates rules as globally reusable abstract units rather than instance-specific facts. In RuleWorld, several scenarios, including single-rule, parallel multi-rule, and multi-hop reasoning, are settled for comprehensive evaluation. We further propose **DynaRule**, an end-to-end framework that injects the given rules into the KV cache and turns retrieval into an internal, learnable, step-wise process. Specifically, DynaRule employs Stacked Step-Level Attention Training with a special `<search>` token to enable dynamic rule re-attention and updating during inference. In this way, the model can re-attend to the most relevant rules at each step, dynamically replacing outdated ones to support more stable multi-step reasoning. Experiments on RuleWorld show that existing LLMs face challenges under large rule pools, while DynaRule improves average QA accuracy by up to 19 points and achieves over 85% Recall@1 at 10K rules, outperforming strong baselines by large margins. We make our code and dataset available here: Beyond-Factual-Knowledge.

1 Introduction

Large language models (LLMs) excel at text understanding, analysis, and generation (OpenAI et al., 2024; Zhao et al., 2026), motivating a substantial body of work that evaluates their knowledge capabilities (Petroni et al., 2019; Lin et al., 2022; Huang and Chang, 2023). However, these evaluations primarily emphasize the capabilities of understanding and applying factual knowledge, while often ignoring procedural knowledge, i.e., knowledge of how to act or reason. In realistic settings, procedural knowledge is maintained as a large repository of reusable procedural entries (e.g., policies, guidelines, or rules), requiring models to localize and compose the relevant items across reasoning steps. In this paper, we focus on rules as a concrete and widely used form of procedural knowledge. To evaluate related capabilities of LLMs on rules, existing benchmarks (Sun et al., 2024b; Han et al., 2024; Qi et al., 2025) typically provide the relevant rules directly as problem-specific premises in each reasoning instance. This design reduces reusable procedural rules to per-question hints and thus ignores the shared nature of rule knowledge across QA instances. As a result, they obscure whether a model can (i) identify the correct reusable rules from a large, shared rule repository for a query and (ii) apply them reliably in multi-step reasoning.

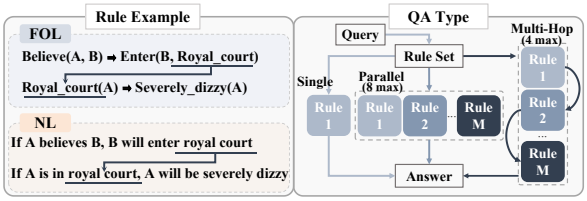


Figure 1: Illustration of rule examples and RuleWorld QA types. Rules from different types may interact, and the benchmark includes three QA types: single-rule, parallel multi-rule, and multi-hop reasoning.

To specifically evaluate whether LLMs could correctly localize and apply rules from large-scale injected rule repository according to the given query, this paper constructs **RuleWorld**, a benchmark designed to measure the rule utilization capability of LLMs. RuleWorld comprises 4.94 million externally provided procedural rules that are ab-

* Corresponding authors.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 9128–9152
October 24-29, 2026 ©2026 Association for Computational Linguistics

stract, non-commonsense, and entity-independent, which prevents models from relying on internalized knowledge. For instance, a rule like "if X is happy, then X will go to the forest" leaves X unspecified, requiring the model to ground the placeholder to the query before applying the rule. Consequently, strong performance requires selecting a small subset of relevant rules from a large rule set and executing them correctly during reasoning. In RuleWorld, the rules are organized into four major types (*Attributes*, *Environments*, *Actions*, and *States*) with seven subtypes, available in both first-order logic (FOL) and natural-language (NL) forms. Crucially, RuleWorld is compositional by design: a rule's conclusion can become the premise of other rules, yielding cross-type interactions (e.g., an *Action* can change the *Environment* and cause downstream changes; Figure 1, left) that pose realistic challenges for rule localization and utilization during reasoning. To systematically assess these capabilities, we define three QA types that capture distinct modes of rule utilization (Figure 1, right). *Single-Rule QA* requires applying exactly one rule to answer a question, providing a basic test of rule grounding and application. *Parallel Multi-Rule QA* involves several sub-questions in one instance. Each sub-question is an independent, non-sequential step that may require multiple rules. The model solves these steps separately, then aggregates the sub-answers into a combined final answer (up to eight gold rules per instance). *Multi-Hop Rule QA* requires chaining reasoning steps, where each hop may involve applying multiple rules, and earlier conclusions become later premises, requiring the model to track intermediate states across the chain (up to four hops). Together, they form eleven sub-tasks by varying the number of gold rules and the number of hops, progressively increasing the difficulty of rule localization and application.

Through detailed evaluations on RuleWorld, we find that there are two major challenges for current LLMs to effectively leverage large injected rule sets. (i) Injected rules are highly abstract and thus weakly match question surface forms, making both rule retrieval and utilization brittle at scale. (ii) The ability of step-wise rule utilization is required under both parallel aggregation and multi-hop state tracking, where the target rules change across steps, making retrieval and reasoning unstable. Existing methods such as retrieval-augmented generation (RAG) (Lewis et al., 2020) are sensitive to retrieval quality and prone to semantic mismatch, while internal injection (Wang et al., 2025; Yu et al., 2026) often exhibits unstable rule selection at scale and degrades substantially in multi-step reasoning.

To this end, we introduce **DynaRule**, which conducts dynamic rule selection and updating entirely within the model's internal space. Inspired by recent work that treats attention as key-value association (Wang et al., 2025; Yu et al., 2026), external rules are encoded with a pretrained sentence encoder, aligned via single-layer adapters, and directly injected into the LLMs' KV cache. In specific, DynaRule is trained in two stages. First, we identify the model's *confidence layer* as the layer at which attention entropy over injected rules is minimized, indicating the most decisive focus on a small set of relevant rules while suppressing irrelevant ones. Second, we perform *Stacked Step-Level Attention Training* on this layer by introducing a special `<search>` token, which guides attention to the rules required at each reasoning step and turns retrieval into an internal, learnable process. During inference, emitting `<search>` triggers rule re-attention and re-retrieval, replacing outdated rule entries in the KV cache with newly selected ones. This design internally aligns abstract rule selection with step-wise decoding via `<search>`-triggered re-attention, stabilizing retrieval and reasoning as the target rules change across steps.

Comprehensive experiments demonstrate that RuleWorld poses substantial challenges to existing LLMs (e.g., DeepSeek V3.2 (DeepSeek-AI et al., 2025)) and GPT-5.5 (OpenAI, 2026). DynaRule consistently outperforms all baselines across diverse rule injection settings, improving average QA accuracy by up to 19 points and achieving gains of up to 48 points on specific sub-tasks. It also achieves over 90% Recall@10 and over 85% Recall@1 at 10K rules, surpassing the strongest baseline by more than 60 points. These results show that step-level dynamic integration can identify the relevant procedural rules at each step and reliably leverage them during reasoning.

Our contributions are two-fold:

- We introduce **RuleWorld**, a large-scale benchmark containing millions of abstract procedural rules in FOL and NL, covering four types, seven sub-types, and eleven QA sub-tasks for evaluating rule-application ability.
- We propose **DynaRule**, an end-to-end rule integration framework with learnable retrieval that injects external rules into the KV cache

and uses <search>-driven step-wise attention to select and update rules, yielding large gains in QA and retrieval accuracy and enabling more reliable procedural reasoning.

2 Related Work

Benchmarks for Logical Reasoning Existing logical reasoning benchmarks usually model rules as per-instance premises rather than as a reusable and globally consistent rule system (Clark et al., 2020; Tafjord et al., 2021; Nakamura et al., 2023; Han et al., 2024; Parmar et al., 2024; Patel et al., 2024; Zhou et al., 2025; Qi et al., 2025). As shown in Table 1 (left), prior benchmarks each cover only part of the space, such as FOL or NL rules, faithful reasoning chains, or controlled composition, but they generally do not combine large rule pool localization with shared rule knowledge across instances. This limits diagnosis of whether models truly retrieve and apply reusable rules, especially beyond aggregate accuracy in true/false or multiple choice settings (Tian et al., 2021; Liu et al., 2023; Parmar et al., 2024; Patel et al., 2024; Zhou et al., 2025; Qi et al., 2025). RuleWorld addresses this gap by introducing a contradiction free shared rule set and controlled evaluation of single-rule, parallel multi-rule, and multi-hop reasoning.

Methods for Rule Reasoning Existing methods for rule reasoning with LLMs mainly fall into three categories. (1) Full fine-tuning internalizes reasoning patterns through large synthetic corpora (Xu et al., 2024; Morishita et al., 2024; Jiang et al., 2025), but is costly and does not explicitly model globally reusable rules. (2) Prompting-based approaches decompose reasoning into multiple stages such as planning, rewriting, or solver based verification (Pan et al., 2023; Sun et al., 2024a; Servantez et al., 2024), yet often rely on strong closed-source models or external tools and remain brittle under difficult rule localization. (3) Retrieval based approaches inject rules at inference time, but both standard RAG (Lewis et al., 2020) and recent KV cache based methods (Wang et al., 2025; Yu et al., 2026) still struggle with reliable step-level rule selection. DynaRule addresses this gap by jointly learning step-level retrieval and reasoning in an end-to-end manner.

3 Preliminaries

Rule Formulation Rules follow a causal structure in which the *conclusion* holds only when the *premise* is met (Brachman and Levesque, 2004). Each rule is represented as a pair $(\mathbf{p}, \mathbf{c})$, with $\mathbf{p}$ defined as the premise and $\mathbf{c}$ as the conclusion. A rule set with M rules is written as $\mathcal{R} = \{(\mathbf{p}_m, \mathbf{c}_m)\}_{m=1}^{M}$.

Rule Knowledge Injection via KV Cache Inspired by (Wang et al., 2025; Yu et al., 2026), we treat attention as a normalized key–value matching mechanism. Under this view, each rule $(\mathbf{p}_m, \mathbf{c}_m)$ is represented as a key–value pair, where the full rule $(\mathbf{p}_m, \mathbf{c}_m)$ acts as the key and the conclusion $\mathbf{c}_m$ as the value. Specifically, we encode each rule using a pretrained sentence encoder to obtain $(\mathbf{k}_m, \mathbf{v}_m) = \textsc{Encode}((\mathbf{p}_m, \mathbf{c}_m), \mathbf{c}_m)$, and project them from the encoder dimension P into the model embedding space D via single-linear adapters at layer l: $(\tilde{\mathbf{k}}_m^l, \tilde{\mathbf{v}}_m^l) = (\mathbf{k}_m \tilde{\mathbf{W}}_K^l, \mathbf{v}_m \tilde{\mathbf{W}}_V^l)$, where $\tilde{\mathbf{W}}_K^l, \tilde{\mathbf{W}}_V^l \in \mathbb{R}^{P \times D}$. This design allows each rule instance to be uniquely identified, even when multiple rules share identical premises. Trained over large-scale rule corpora, layer-specific adapters learn robust mappings that enable rule embeddings to be smoothly integrated into the attention mechanism via rectangular attention. At each layer, the projected rule representations with M entries are directly inserted into the corresponding KV cache, which contains N contextual key-value pairs $\mathbf{K}^l, \mathbf{V}^l \in \mathbb{R}^{N \times D}$, resulting in expanded representations $\hat{\mathbf{K}}^l, \hat{\mathbf{V}}^l \in \mathbb{R}^{(M+N) \times D}$. Formally:

$$\hat{\mathbf{K}}^l = \left[\, \tilde{\mathbf{K}}^l \;\; \mathbf{K}^l \,\right], \quad \hat{\mathbf{V}}^l = \left[\, \tilde{\mathbf{V}}^l \;\; \mathbf{V}^l \,\right], \quad (1)$$

where $\tilde{\mathbf{K}}^l = [\tilde{\mathbf{k}}_1^l, \ldots, \tilde{\mathbf{k}}_M^l], \tilde{\mathbf{V}}^l = [\tilde{\mathbf{v}}_1^l, \ldots, \tilde{\mathbf{v}}_M^l]$. To obtain auxiliary queries, we use a dedicated projection adapter $\tilde{\mathbf{W}}_Q^l$ that transforms the hidden states $\mathbf{X}^l$ into a new query matrix $\tilde{\mathbf{Q}}^l = \mathbf{X}^l \tilde{\mathbf{W}}_Q^l \in \mathbb{R}^{N \times D}$. In contrast, the original query matrix $\mathbf{Q}^l$ is retained for standard contextual self-attention. Since only the key–value sequence length is modified, the output dimensionality remains identical to standard self-attention, allowing rule information to be seamlessly integrated into the hidden states. The resulting attention computation is:

$$\text{Attention} = \text{Softmax}\left(\left[\frac{\tilde{\mathbf{Q}}^l \left(\tilde{\mathbf{K}}^l\right)^\top}{\sqrt{D}} \;\middle|\; \frac{\mathbf{Q}^l \left(\mathbf{K}^l\right)^\top}{\sqrt{D}} \right] \right) \hat{\mathbf{V}}^l. \quad (2)$$

4 RuleWorld Benchmark

RuleWorld is designed to evaluate whether LLMs can localize and apply reusable procedural rules

Benchmark	FOL Rule	NL Rule	Faithful Natural Language Reasoning Chain	Controlled Composition Taxonomy	Large Shared Rule Pool with Localization
RuleTaker (Clark et al., 2020)	✗	✓	✗	⊖	✗
ProofWriter (Tafjord et al., 2021)	✗	✓	✓	⊖	✗
ProntoQA (Saparov and He, 2023)	✓	✓	✓	⊖	✗
FOLIO (Han et al., 2024)	✓	✓	✗	⊖	✗
LogicBench (Parmar et al., 2024)	⊖	✓	✗	✓	✗
Multi-LogiEval (Patel et al., 2024)	⊖	✓	✗	✓	✗
RuleArena (Zhou et al., 2025)	✗	✓	⊖	⊖	✓
ProverQA (Qi et al., 2025)	✓	✓	✓	✓	✗
RuleWorld (Ours)	✓	✓	✓	✓	✓

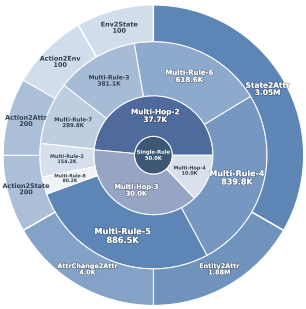


Table 1: **Left**: Comparison of rule-guided reasoning benchmarks across five dimensions. ✓ = Yes, ✗ = No, ⊖ = Partial. **Right**: Distribution of 3.37M QA instances over 11 sub-tasks; the outer ring shows seven rule sub-types, with thick separators marking major types. Suffixes denote gold rule counts or hops.

from a large shared repository, yielding a large-scale, rule-centric QA benchmark with 3.37M instances, as shown in Table 1 (right). RuleWorld comprises 4.94 million rules in both FOL and NL forms that are globally consistent, abstract, intentionally non-commonsense (e.g., "*entering a forest sets an entity on fire*"), and conflict-free, making the same rules reusable across all QA instances without relying on prior knowledge. The rules are organized into four major types and seven sub-types, where the conclusion of one rule can become the premise of another, enabling cross-rule interactions. Detailed data statistics, the full rule and QA construction pipeline, and complete examples are provided in Appendices A.1, A.2, and A.3, respectively. Moreover, we categorize RuleWorld into three task types, each targeting a distinct dimension of rule application:

(1) Single-Rule QA (50K instances). Each instance requires applying exactly one rule to derive the correct answer. This setting serves as the simplest form of rule grounding and application.

(2) Parallel Multi-Rule QA (3.25M instances). Each instance contains several parallel sub-questions, which the model answers independently and then combines the resulting conclusions into a final answer. Difficulty scales with the number of gold rules, up to 8 per instance.

(3) Multi-Hop Rule QA (77.7K instances). Each instance requires sequential reasoning, where the conclusion of one step serves as the premise for the next, necessitating tracking of intermediate states across the chain. Difficulty scales with the number of hops, up to 4 per instance.

Together, these tasks systematically evaluate LLMs' ability to localize and apply rules across 11 sub-tasks by varying gold-rule and hop counts.

5 DynaRule

5.1 Training Process

We adopt a two-stage scheme: (1) identify the model's *confidence layer* via layer-wise attention entropy; (2) introduce a `<search>` token and train this layer with a Stacked Step-Level Attention Loss for step-wise rule retrieval (Figure 2, left). Rule attention uses queries projected by dedicated $\tilde{\mathbf{W}}_Q^l$. We freeze the base model in both stages and update $\tilde{\mathbf{W}}_Q^l$, $\tilde{\mathbf{W}}_K^l$, and $\tilde{\mathbf{W}}_V^l$, additionally unfreezing the embedding layer in Stage 2 to learn `<search>`.

Confidence Layer Identification Not all layers are equally informative for rule selection when external rule knowledge is injected into the KV cache. In particular, we observe that certain layers exhibit significantly sharper attention distributions over rule keys, indicating that rule relevance is most clearly differentiated at these depths. To quantify this effect, let the softmax-normalized attention weights over the M injected rule keys at layer l, given a query q, be $\{\alpha_m^l(q)\}_{m=1}^M$, and define the corresponding attention entropy as $H^l(q) = -\sum_{m=1}^M \alpha_m^l(q) \log \alpha_m^l(q)$. We identify the *confidence layer* as the layer with the lowest average attention entropy across the query set, i.e., $\arg\min_l \mathbb{E}_{q\sim\mathcal{D}_q}\left[H^l(q)\right]$, as this layer concentrates attention on a small subset of relevant rules and provides the most decisive and stable rule relevance signals for guiding step-level rule retrieval.

Stacked Step-Level Attention Training Objective We supervise the *confidence layer*'s rule attention to enable efficient step-wise rule retrieval at inference. During training, we use the full teacher-forced sequence (question plus answer), with the answer divided into T reasoning steps. We treat step $t = 1$ as purely question-conditioned retrieval, since the question fully specifies the retrieval in-

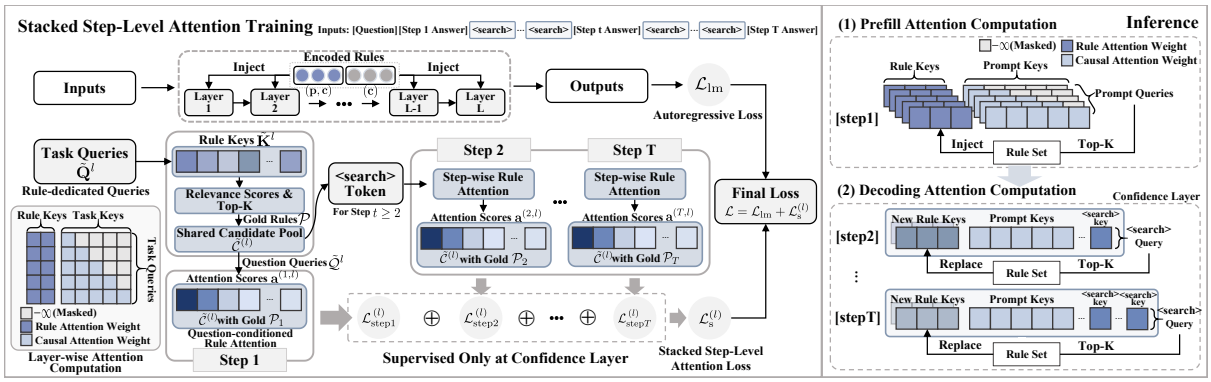


Figure 2: Overview of Stacked Step-Level Attention Training (**left**) and dynamic rule attention at inference (**right**). Training builds a shared candidate pool $\tilde{\mathcal{C}}^{(l)}$ from sequence-level relevance between $\tilde{\mathbf{Q}}^l$ and injected rule keys $\tilde{\mathbf{K}}^l$, and learns step-level retrieval with question and <search> tokens. At inference, top-K rules are selected at prefill and updated through re-attention triggered by <search> tokens. Values are omitted for clarity.

tent. For $t \geq 2$, we insert a <search> token before the step-t answer content, and use it as an explicit retrieval trigger to support step-wise retrieval conditioned on intermediate reasoning states. To align training with inference-time per-layer pruning and to expose the *confidence layer* to hard candidates, we first construct a stable Top-K candidate set at each layer. Specifically, at each layer l, we compute a *sequence-level* relevance score for each rule by averaging full-sequence token queries $\tilde{\mathbf{Q}}^l = \{\tilde{\mathbf{q}}_n^l\}_{n=1}^N$ against injected rule keys $\tilde{\mathbf{K}}^l = \{\tilde{\mathbf{k}}_m^l\}_{m=1}^M$, i.e., $\mathbf{s}_m^{(l)} = \frac{1}{N}\sum_{n=1}^N (\tilde{\mathbf{q}}_n^l)^\top \tilde{\mathbf{k}}_m^l$, and retain the Top-$K$ index set as $\mathcal{C}^{\text{top},(l)}$. This token-averaged scoring makes Top-K selection less sensitive to step boundaries, yielding a stable candidate set under batched training, while explicit supervision is only applied at the *confidence layer*.

$$\tilde{\mathcal{C}}^{(l)} = \textsc{KeepDim}\left(\underbrace{\textsc{TopK}(\{\mathbf{s}_m^{(l)}\}_{m=1}^M, K)}_{\mathcal{C}^{\text{top},(l)}} \cup \left(\bigcup_{t=1}^T \mathcal{P}_t\right),\ K\right) \quad (3)$$

Let $\mathcal{P}_t$ denote the gold rule indices required at each step t. As shown in Eq. (3), at the *confidence layer* we augment the Top-K set with the gold indices from all steps, i.e., $\bigcup_{t=1}^T \mathcal{P}_t$, while removing an equal number of other indices to maintain dimensional consistency via a $\textsc{KeepDim}(\cdot)$ operation, yielding a shared candidate index pool $\tilde{\mathcal{C}}^{(l)}$ across steps. This prevents pruning away future-step gold indices and enables cross-step discrimination: at step t, rules indexed by $\mathcal{P}_t$ are positives, while the remaining rules in $\tilde{\mathcal{C}}^{(l)}$, including gold rules from other steps, act as hard negatives.

$$\mathbf{a}_m^{(t,l)} = \begin{cases} \dfrac{1}{\sqrt{D}} \dfrac{1}{|\tilde{\mathcal{Q}}^l|} \displaystyle\sum_{p \in \tilde{\mathcal{Q}}^l} (\tilde{\mathbf{q}}_p^l)^\top \tilde{\mathbf{k}}_m^l, & t = 1, \\ \dfrac{1}{\sqrt{D}} (\tilde{\mathbf{q}}_{\texttt{<search>},t}^l)^\top \tilde{\mathbf{k}}_m^l, & t \geq 2. \end{cases} \quad (4)$$

We then apply Eq. (4) to each rule $m \in \tilde{\mathcal{C}}^{(l)}$ to compute an attention score $\mathbf{a}_m^{(t,l)}$, which serves as the retrieval signal for step-conditioned rule selection. For step $t = 1$, retrieval is question-conditioned; thus we compute $\mathbf{a}_m^{(1,l)}$ by averaging dot products between the rule key $\tilde{\mathbf{k}}_m^l$ and the question-token queries indexed by $\tilde{\mathcal{Q}}^l$. For subsequent steps $t \geq 2$, retrieval intent depends on intermediate reasoning states rather than the static question; therefore we use the query state of the step-specific <search> token inserted before the answer at step t, denoted as $\tilde{\mathbf{q}}_{\texttt{<search>},t}^l$, to form a step-adaptive query for rule identification. Since single-step QA requires only one step, no <search> token is involved.

$$\mathcal{L} = \underbrace{\left(-\frac{1}{N}\sum_{n=1}^N \log p_\theta(x_n \mid x_{<n})\right)}_{\mathcal{L}_{\text{lm}}\text{: Autoregressive Language Modeling Loss}} + \underbrace{\left(-\frac{1}{\sum_{t=1}^T |\mathcal{P}_t|}\sum_{t=1}^T \sum_{i \in \mathcal{P}_t} \log \frac{\exp(\mathbf{a}_i^{(t,l)}/\mathcal{T})}{\sum_{j \in \tilde{\mathcal{C}}^{(l)} \setminus (\mathcal{P}_t \setminus \{i\})} \exp(\mathbf{a}_j^{(t,l)}/\mathcal{T})}\right)}_{\mathcal{L}_{\text{s}}^{(l)}\text{: Stacked Step-Level Attention Loss at layer } l} \quad (5)$$

Next, for each step, we apply a cross-entropy loss per positive rule over the candidate pool $\tilde{\mathcal{C}}^{(l)}$, where other positives at the same step are masked out, encouraging higher attention on the corresponding rule against hard negatives. A temperature coefficient $\mathcal{T}$ is introduced to sharpen the contrast between scores, amplifying differences in rule relevance. Finally, the losses from all steps are stacked, forming a Stacked Step-Level Attention Loss $\mathcal{L}_{\text{s}}^{(l)}$ that jointly supervises correct rule usage across the entire reasoning chain. This formulation transforms the retrieval of external abstract rules

into an internal, learnable, and fully differentiable process. The final loss, as shown in Eq. (5), is $\mathcal{L} = \mathcal{L}_{\text{lm}} + \mathcal{L}_{\text{s}}^{(l)}$, where $\mathcal{L}_{\text{lm}}$ is the standard autoregressive language modeling loss.

5.2 Inference Process

During inference, the model performs rule retrieval, updating, and reasoning in a fully end-to-end manner within its internal intent space (Figure 2, right). Inference consists of a prefill stage followed by decoding. In the prefill stage, attention over injected rule keys and Top-K selection are computed at all layers, while explicit rule selection and KV cache updating are carried out only at the *confidence layer*. The model has been trained to emit the special <search> token at the points where additional rule retrieval is required, since multi-step training answers include supervised <search> insertions. Accordingly, if the question requires only one step, decoding proceeds directly to the final answer without emitting <search>. For multi-step cases, whenever the decoder outputs <search>, the hidden state of this token is used as a new query to recompute attention over the rule set at the *confidence layer*, and the newly selected rules replace the previous rules in the KV cache, enabling step accurate grounding throughout the reasoning chain.

6 Experiments

6.1 Experiment Settings

We provide the detailed training and evaluation implementation settings in Appendix B.

Model Selection We adopt Qwen2.5-7B-Instruct (Qwen et al., 2025) as the base model and Qwen3-Embedding-8B (Yang et al., 2025) as the text encoder. To analyze the stability of the confidence layer across model architectures and scales, we further evaluate Llama-3-8B-Instruct (Grattafiori et al., 2024), smaller Qwen2.5 models (1.5B and 3B) (Qwen et al., 2025), and bge-m3 (Chen et al., 2025). For strong prompting baselines, we include larger Qwen2.5 models (14B, 32B, 72B) (Qwen et al., 2025), Qwen3-32B (Yang et al., 2025), Llama3 variants (Llama-3.1-70B-Instruct, Llama-3.3-70B-Instruct) (Grattafiori et al., 2024), deepseek-chat (DeepSeek V3.2) (DeepSeek-AI et al., 2025), gpt-5.1-2025-11-13 (OpenAI, 2025), gpt-5.5-2026-04-23 (OpenAI, 2026), and claude-sonnet-4-6 (Anthropic, 2026).

Baselines We compare our approach against the following representative methods:

(1) Prompting (full-context rule injection). All rules are prepended to the prompt, with GPU memory limiting injection to 2000 rules for the 7B model and 1000 for larger models.

(2) RAG (Lewis et al., 2020). An external retriever selects top-K rules, which are prepended to the prompt. Retrieval is performed once before generation, with no updates during reasoning.

(3) KBLaM (Wang et al., 2025). A state-of-the-art end-to-end method that integrates external knowledge by projecting key–value pairs into the model's internal KV cache space.

(4) SR-KI (Yu et al., 2026). A novel end-to-end method that supervises retrieval through an attention-based retrieval loss in the KV cache, but does not explicitly decompose multi-step reasoning, leading to a single-step retrieval signal.

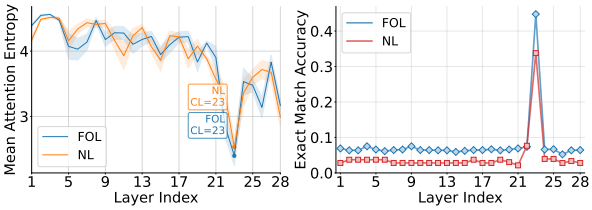


Figure 3: Confidence layer (CL) identification (FOL/NL), with shading showing attention entropy standard deviation (**left**). Per-layer retention analysis with correct rules injected into a single target layer (**right**).

6.2 Confidence Layer Identification

We identify Qwen2.5-7B-Instruct's confidence layer by injecting 100 rules into the KV cache under both FOL and NL representations (Figure 3, left). In both cases, attention entropy is minimized at layer 23, showing that the same confidence layer emerges regardless of representation. Additional identification results across settings are reported in Appendix C.1. To further assess the functional importance of this layer, we run a per-layer retention experiment with both FOL and NL rules: the 100 injected rules are correct only at one target layer, while all other layers receive randomly sampled incorrect rules. As shown in Figure 3 (right), accuracy peaks when correct rules are retained at the identified confidence layer for both representations. This result highlights the critical role of the confidence layer and validates it as the base layer for applying Stacked Step-Level Attention Training.

Method	First-order Logic						Natural Language						Avg.
	Single	PM (2–4)	PM (5–8)	MH-2	MH-3	MH-4	Single	PM (2–4)	PM (5–8)	MH-2	MH-3	MH-4	
Rule Num = 100													
Prompting	0.8200	0.7389	0.4917	0.5000	0.0600	0.0400	0.8200	0.8194	0.5684	0.3400	0.1000	0.1200	0.4515
KBLaM	0.9800	**0.9939**	**0.9871**	**1.0000**	0.9000	0.9400	**1.0000**	0.9917	**0.9863**	**0.9600**	0.4800	**0.7200**	0.9116
SR-KI	**1.0000**	0.9883	0.9750	0.9800	0.9200	0.9000	**1.0000**	0.9750	0.9583	**0.9600**	0.5400	0.5600	0.8964
DynaRule	**1.0000**	0.9917	0.9863	0.9600	**0.9300**	**0.9500**	**1.0000**	**0.9967**	0.9825	0.9200	**0.9600**	0.7000	**0.9481**
Rule Num = 1000													
Prompting	0.8000	0.6211	0.2654	0.2800	0.0400	0.0000	0.6200	0.6472	0.2792	0.0600	0.0200	0.0000	0.3027
RAG_{dense}	0.6600	0.3589	0.2029	0.0400	0.0200	0.0200	0.8800	0.5750	0.2383	0.0200	0.0000	0.0000	0.2513
RAG_{bm25}	0.5200	0.5483	0.2825	0.0000	0.0000	0.0000	0.5800	0.3772	0.2379	0.0000	0.0000	0.0000	0.2122
RAG_{hybrid}	0.6200	0.6044	0.3429	0.0000	0.0000	0.0200	0.8200	0.5300	0.3154	0.1600	0.0000	0.0200	0.2861
KBLaM	0.9000	0.8639	0.8888	0.8400	0.4200	0.4400	0.7800	0.8272	0.8729	0.6800	0.3400	0.4200	0.6894
SR-KI	0.9800	0.9483	0.8779	0.7800	0.2200	0.2800	**0.9800**	0.8767	0.8263	0.5800	0.1400	0.0400	0.6274
DynaRule	**1.0000**	**0.9931**	**0.9729**	**0.8500**	**0.6250**	**0.6500**	**0.9800**	**0.9833**	**0.9642**	**0.8400**	**0.6400**	**0.6800**	**0.8482**
Rule Num = 10000													
RAG_{dense}	0.4000	0.2411	0.1088	0.0000	0.0400	0.0000	0.6600	0.2744	0.0842	0.0000	0.0000	0.1000	0.1590
RAG_{bm25}	0.3400	0.1983	0.0950	0.0200	0.0000	0.0000	0.6400	0.2817	0.1638	0.0000	0.0000	0.0400	0.1482
RAG_{hybrid}	0.4400	0.4022	0.1800	0.0000	0.0200	0.0000	0.6000	0.4072	0.2158	0.0200	0.0000	0.0400	0.1938
KBLaM	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
SR-KI	0.7400	0.6194	0.4946	0.3800	**0.2000**	0.1200	0.7400	0.5133	0.4071	0.1200	0.0800	0.0400	0.3712
DynaRule	**0.9200**	**0.9211**	**0.7904**	**0.5200**	0.1600	**0.1400**	**0.8000**	**0.8600**	**0.7288**	**0.6000**	**0.1600**	**0.2400**	**0.5700**

Table 2: Exact match accuracy on RuleWorld with Qwen2.5-7B-Instruct. **Single**: Single-Rule QA. **PM**: Parallel Multi-Rule QA, where columns 2–4 and 5–8 report mean exact match over the corresponding rule counts. **MH**: Multi-Hop Rule QA, where the numeric suffix denotes hop count. RAG_{dense} and RAG_{bm25} use Qwen3-Embedding-8B and BM25 retrievers, respectively; RAG_{hybrid} uses RRF (Cormack et al., 2009) with a fusion constant of 60. Detailed results for additional rule sizes are reported in Tables 8 and 9.

6.3 Main Results

6.3.1 Experiments on Rule Reasoning

We use exact match accuracy as the evaluation metric. Table 2 presents the main results on both FOL and NL rules, from which we draw two key observations. Detailed results under additional rule size settings are provided in Tables 8 and 9. (1) Prompting only works at very small scales and fails on parallel and multi-hop settings, and RAG performs worse due to semantic mismatch. Existing end-to-end knowledge injection methods do not scale: KBLaM and SR-KI are competitive at small scales but degrade sharply as the number of rules increases, especially on parallel and multi-hop QA. (2) Our method consistently performs best across rule scales, exceeding the strongest baseline by up to 19.88 points on average and staying strong under 10000 rules on parallel multi-rule QA (e.g., 0.72+ on multi-rule (5–8), >1.5× relative gain). Multi-hop QA poses a greater challenge than parallel settings, as it relies on rule chaining. We further report prompting results of strong LLMs under a 1000-rule setting in Table 3, while detailed results and case studies are shown in Appendices C.2 and C.3. Even these strong models remain inferior to DynaRule, which achieves better overall performance.

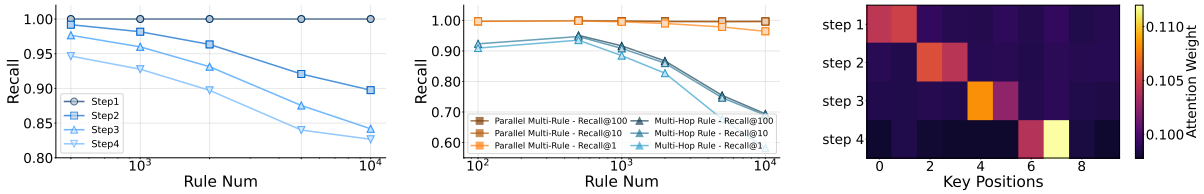
Method	First-order Logic			Natural Language			Avg.
	S-Rule	PM-Rule	MH-Rule	S-Rule	PM-Rule	MH-Rule	
Qwen2.5-14B-Instruct	0.7200	0.4486	0.1933	0.6200	0.4031	0.1933	0.4297
Qwen2.5-32B-Instruct	0.8400	0.5038	0.2067	0.7400	0.5357	0.2200	0.5077
Qwen2.5-72B-Instruct	0.7200	0.5945	0.2533	0.7600	0.6698	0.1867	0.5307
Llama-3.1-70B-Instruct	0.8600	0.5571	0.2067	0.9600	0.5633	0.1467	0.5490
Llama-3.3-70B-Instruct	0.8600	0.4555	0.1667	0.9600	0.4686	0.1867	0.5162
Qwen3-32B	0.5800	0.5326	0.2600	0.8800	0.4933	0.3333	0.5132
deepseek-chat	0.8800	0.7821	0.3467	0.9000	0.7236	0.3800	0.6687
gpt-5.1-2025-11-13	0.9600	0.7576	0.3200	**1.0000**	0.7602	0.2800	0.6796
gpt-5.5-2026-04-23	0.9800	0.8593	0.3533	0.8000	0.9107	0.4667	0.7283
claude-sonnet-4-6	0.9000	0.9117	0.5133	0.8300	0.8595	0.6433	0.7763
DynaRule	**1.0000**	**0.9816**	**0.7083**	0.9800	**0.9724**	**0.7200**	**0.8937**

Table 3: Partial prompting comparison results on RuleWorld under 1000 rules. **S-Rule**: Single-Rule QA; **PM-Rule**: Parallel Multi-Rule QA; **MH-Rule**: Multi-Hop Rule QA. Detailed results are provided in Appendix C.2.

6.3.2 Experiments on Rule Retrieval

Table 4 reports main retrieval performance on FOL and NL rules, where Recall@1 sets K to the number of gold rules per instance or step. Table 12 further provides results for additional rule size settings. For DynaRule, we report step-level correctness averaged across steps. We observe two findings. (1) Baselines such as dense retrieval, BM25 and hybrid search are limited by semantic mismatch; KBLaM fails due to the lack of retrieval supervision, and SR-KI's single-step retrieval paradigm cannot capture interacting rules. In contrast, DynaRule performs best across all settings, beating the strongest baseline by up to 61.98 (FOL) and 53.42 (NL) points. (2) Multi-step retrieval compounds errors. As shown in Figures 4a and 4b, DynaRule's recall drops at deeper steps, and multi-hop underperforms parallel multi-rule at the same scale, consistent with early-step mistakes propagating and leading to weaker performance in multi-hop settings.

The rule reasoning and retrieval results are stable. Detailed seed-level standard deviations and

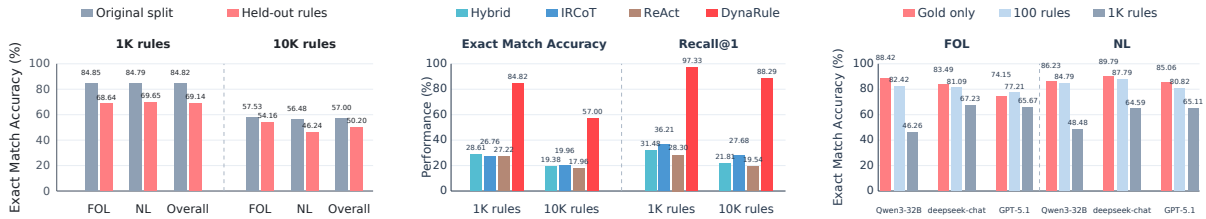


(a) DynaRule Recall@100 across reasoning steps under FOL.

(b) DynaRule recall under FOL in parallel multi-rule and multi-hop settings.

(c) Rule attention weights (4-step). Required rules are ordered for clarity.

Figure 4: Retrieval performance (FOL) across QA types and reasoning steps, with step-wise attention visualization.



(a) Exact match accuracy on original and held-out rules.

(b) Exact match accuracy and Recall@1 results compared with Iterative RAG.

(c) Exact match accuracy under only the gold rules, 100 rules, and 1K rules.

Figure 5: Rule generalization, iterative retrieval, and gold-rule application analyses.

Method	First-order Logic			Natural Language		
	Recall@100	Recall@10	Recall@1	Recall@100	Recall@10	Recall@1
Rule Num = 100						
Dense	-	0.5325	0.4448	-	0.6646	0.5414
BM25	-	0.6901	0.4375	-	0.5869	0.5292
Hybrid	-	0.6364	0.4793	-	0.5972	0.5128
KBLaM	-	0.5151	0.4122	-	0.5278	0.4025
SR-KI	-	0.9100	0.8000	-	0.8963	0.7768
DynaRule	-	**0.9773**	**0.9733**	-	**0.9909**	**0.9857**
Rule Num = 1000						
Dense	0.5529	0.3635	0.3028	0.6961	0.4490	0.3587
BM25	0.7559	0.2962	0.1887	0.5946	0.4892	0.4424
Hybrid	0.7935	0.3561	0.2456	0.7618	0.4464	0.3839
KBLaM	0.5237	0.3532	0.1739	0.5389	0.3351	0.1686
SR-KI	0.9331	0.7369	0.5318	0.9288	0.7213	0.5429
DynaRule	**0.9762**	**0.9737**	**0.9656**	**0.9932**	**0.9892**	**0.9810**
Rule Num = 10000						
Dense	0.3710	0.2573	0.2227	0.4619	0.2940	0.2241
BM25	0.3017	0.1102	0.0418	0.4970	0.4058	0.3703
Hybrid	0.5023	0.2344	0.1338	0.5876	0.3559	0.3023
KBLaM	0.3884	0.0642	0.0239	0.3554	0.0776	0.0282
SR-KI	0.7758	0.4180	0.2415	0.7539	0.4465	0.2898
DynaRule	**0.9145**	**0.9130**	**0.8613**	**0.9614**	**0.9526**	**0.9045**

Table 4: Retrieval performance on FOL and NL rules, with extended results in Table 12.

t-based 95% confidence intervals are reported in Appendix C.4. Detailed step-wise and QA-type analysis are provided in Appendix C.5.

6.4 Analysis

Step-Aware Retrieval Analysis Figure 4c shows 4-step rule retrieval, where each step contains two gold rules. In Step 1, attention focuses on Rules 0–1 (the gold rules for Step 1). In Step 2, it shifts to Rules 2–3 (the gold rules for Step 2) while attention to Rules 0–1 drops. Later steps follow the same pattern, indicating targeted, step-aware retrieval triggered by <search>. Detailed attention visualizations across steps are provided in Appendix C.6.

Generalization to Unseen Rules We additionally evaluate the same trained DynaRule models on QA instances constructed from a rule repository disjoint from the training rule subset, without any further adaptation. The held-out setting follows the same setting as the main experiments. In Figure 5a, Overall denotes the aggregate across FOL and NL results. Performance decreases on unseen rules, but remains substantial: at 10K rules, the overall exact match accuracy is 50.20%, only 6.80 percentage points below the original split. This result indicates that the adapters learn a transferable alignment from rule embeddings to the model's internal knowledge state rather than simply memorizing the training repository.

Comparison with Iterative RAG To test whether stronger inference-time retrieval closes the gap, we compare DynaRule with IRCoT (Trivedi et al., 2023) and a ReAct-style (Yao et al., 2023) retrieval agent implemented with LangGraph (LangChain Inc., 2024), using Qwen2.5-7B-Instruct and the same Hybrid RAG retriever as in the main experiments (BM25 and Qwen3-Embedding-8B combined with RRF, $k = 60$). Each retrieval step returns 100 rules. The final answer is generated from the final top-100 aggregated rules, which are also used to compute Recall@1. IRCoT and ReAct use a four-round/loop budget. All comparisons follow the same evaluation setting as the main experiments. Figure 5b reports exact

match accuracy (left) and Recall@1 (right), each averaged across the FOL and NL task results. The results show that merely adding multiple inference-time retrieval steps does not necessarily yield better rule localization and application. At 10K rules, IRCoT and ReAct achieve 19.96%/17.96% exact match accuracy and 27.68%/19.54% Recall@1, respectively, both far lower than DynaRule's 57.00% exact match accuracy and 88.29% Recall@1. DynaRule is more reliable because its internal step-level retrieval is jointly trained with reasoning, enabling rule localization and rule utilization to be aligned with the decoding process.

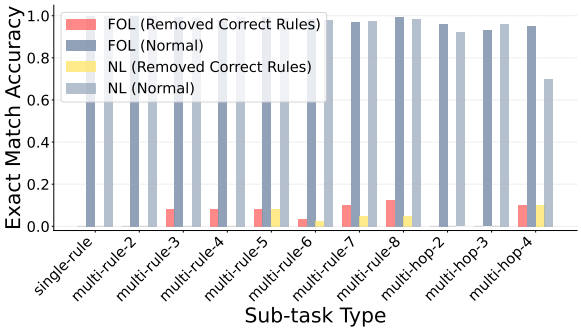
Gold-Rule-Only Analysis Figure 5c compares average exact match accuracy over 11 QA sub-tasks under 100-rule, 1K-rule, and gold-rule-only prompting for Qwen3-32B, deepseek-chat, and gpt-5.1-2025-11-13 (GPT-5.1). Increasing the pool from 100 to 1K lowers accuracy across models and representations, highlighting irrelevant-rule noise. Providing only gold rules removes this noise and isolates rule application: the models attain 74.15–89.79% exact match accuracy across FOL and NL, indicating that the main challenge is accurate rule localization and application.

Rule Dependence Analysis To verify genuine reliance on injected rules, we conduct the ablation study in Figure 6a. With 100 injected rules, removing all correct rules from every layer collapses accuracy under both FOL and NL, often to zero. This shows the model relies on injected rules, and suggests that the linear adapters primarily learn rule alignment rather than memorizing specific rules.

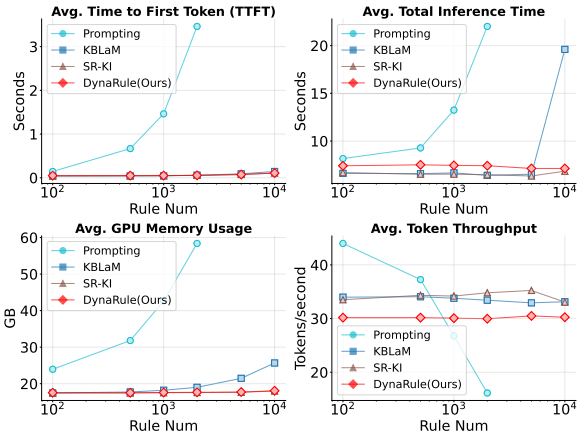
Efficiency Analysis Efficiency is measured by TTFT, total inference time, GPU memory, and token throughput, as shown in Figure 6b. KV cache injection scales as $O((M+N)N)$, far cheaper than quadratic prompt injection with long rule prompts. This reduces TTFT, inference time, and memory, and improves throughput under large rule sets. Moreover, <search> is a single token, so each step adds only $O(M)$ retrieval cost, incurring minimal latency and only a slight drop in throughput.

7 Conclusion

We introduce **RuleWorld**, a large-scale benchmark built from a globally consistent rule set for evaluating LLMs' ability to use external procedural rules in single-rule, parallel multi-rule and multi-hop



(a) Comparison between removing correct rules and the normal setting.



(b) Efficiency analysis of TTFT (top left), inference time (top right), GPU memory usage (bottom left), and token throughput (bottom right) on a single A800 GPU, using Qwen-2.5-7B-Instruct as the base model.

Figure 6: Effectiveness and efficiency evaluation.

settings. We further propose **DynaRule**, a step-level rule integration framework that unifies rule retrieval, updating, and reasoning entirely within the model's internal space. Extensive experiments show that DynaRule substantially improves the reliability and accuracy of procedural rule application. This work exposes the gap between parametric knowledge and explicit rule grounding, advancing more dynamic reasoning.

Limitations

Although RuleWorld provides globally consistent non-commonsense rules and supports diverse reasoning settings, its rules are currently limited to FOL and natural language forms. Future work may explore richer representations, such as executable code or more complex rule-triggering patterns. DynaRule enables end-to-end rule retrieval, updating, and reasoning, but embedding-only rule representations inevitably lose information, which may cause retrieval or application errors as rule pools scale. While we evaluate up to 10,000 injected rules, the

full rule set contains millions, and LLMs already degrade at the 1,000-rule scale. Future work may improve scalability through jointly trained rule encoders, hierarchical or clustered indexing, layer-wise adaptive KV-cache allocation (Yu and Chai, 2025), and more structured rule representations. Future work could also explore language-agnostic knowledge localization, reasoning-oriented representation tuning, structured agent memory, and reasoning-consistent knowledge editing (Cao et al., 2024; Liang et al., 2025, 2026; Zhang et al., 2026).

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No.62406321) and CIPS-SMP-Zhipu Large Model Fund.

References

Anthropic. 2026. Claude sonnet 4.6 system card. Technical report, Anthropic.

Ronald Brachman and Hector Levesque. 2004. *Knowledge representation and reasoning*. Elsevier.

Pengfei Cao, Yuheng Chen, Zhuoran Jin, Yubo Chen, Kang Liu, and Jun Zhao. 2024. One mind, many tongues: A deep dive into language-agnostic knowledge neurons in large language models. *Preprint*, arXiv:2411.17401.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2025. M3-embedding: Multilinguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *Preprint*, arXiv:2402.03216.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2020. Transformers as soft reasoners over language. *Preprint*, arXiv:2002.05867.

Gordon V. Cormack, Charles L A Clarke, and Stefan Buettcher. 2009. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, page 758–759, New York, NY, USA. Association for Computing Machinery.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, and 245 others. 2025. Deepseek-v3.2: Pushing the frontier of open large language models. *Preprint*, arXiv:2512.02556.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, Lucy Sun, Alex Wardle-Solano, Hannah Szabo, Ekaterina Zubova, Matthew Burtell, Jonathan Fan, Yixin Liu, Brian Wong, Malcolm Sailor, and 16 others. 2024. Folio: Natural language reasoning with first-order logic. *Preprint*, arXiv:2209.00840.

Jie Huang and Kevin Chen-Chuan Chang. 2023. Towards reasoning in large language models: A survey. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 1049–1065, Toronto, Canada. Association for Computational Linguistics.

Jin Jiang, Yuchen Yan, Yang Liu, Jianing Wang, Shuai Peng, Xunliang Cai, Yixin Cao, Mengdi Zhang, and Liangcai Gao. 2025. LogicPro: Improving complex logical reasoning via program-guided learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 26200–26218, Vienna, Austria. Association for Computational Linguistics.

LangChain Inc. 2024. Langgraph. `https://github.com/langchain-ai/langgraph`.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Sirui Liang, Pengfei Cao, Jian Zhao, Cong Huang, Jun Zhao, and Kang Liu. 2025. Bias-restrained prefix representation finetuning for mathematical reasoning. *Preprint*, arXiv:2511.10707.

Sirui Liang, Pengfei Cao, Jian Zhao, Wenhao Teng, Xiangwen Liao, Jun Zhao, and Kang Liu. 2026. Learning how to remember: A meta-cognitive management method for structured and transferable agent memory. *Preprint*, arXiv:2601.07470.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252, Dublin, Ireland. Association for Computational Linguistics.

Hanmeng Liu, Jian Liu, Leyang Cui, Zhiyang Teng, Nan Duan, Ming Zhou, and Yue Zhang. 2023. Logiqa 2.0—an improved dataset for logical reasoning in

natural language understanding. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 31:2947–2962.

Terufumi Morishita, Gaku Morio, Atsuki Yamaguchi, and Yasuhiro Sogawa. 2024. Enhancing reasoning capabilities of llms via principled synthetic logic corpus. In *Advances in Neural Information Processing Systems*, volume 37, pages 73572–73604. Curran Associates, Inc.

Mutsumi Nakamura, Santosh Mashetty, Mihir Parmar, Neeraj Varshney, and Chitta Baral. 2023. LogicAttack: Adversarial attacks for evaluating logical consistency of natural language inference. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 13322–13334, Singapore. Association for Computational Linguistics.

OpenAI. 2025. Gpt-5.1 system card. Technical report, OpenAI.

OpenAI. 2026. Gpt-5.5 system card. Technical report, OpenAI.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. 2023. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3806–3824, Singapore. Association for Computational Linguistics.

Mihir Parmar, Nisarg Patel, Neeraj Varshney, Mutsumi Nakamura, Man Luo, Santosh Mashetty, Arindam Mitra, and Chitta Baral. 2024. Logicbench: Towards systematic evaluation of logical reasoning ability of large language models. *Preprint*, arXiv:2404.15522.

Nisarg Patel, Mohith Kulkarni, Mihir Parmar, Aashna Budhiraja, Mutsumi Nakamura, Neeraj Varshney, and Chitta Baral. 2024. Multi-logieval: Towards evaluating multi-step logical reasoning ability of large language models. *Preprint*, arXiv:2406.17169.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, Hong Kong, China. Association for Computational Linguistics.

Chengwen Qi, Ren Ma, Bowen Li, He Du, Binyuan Hui, Jinwang Wu, Yuanjun Laili, and Conghui He. 2025. Large language models meet symbolic provers for logical reasoning evaluation. In *The Thirteenth International Conference on Learning Representations*.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. *Preprint*, arXiv:1910.02054.

Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. 1994. Okapi at trec-3. In *Text Retrieval Conference*.

Abulhair Saparov and He He. 2023. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. *Preprint*, arXiv:2210.01240.

Sergio Servantez, Joe Barrow, Kristian Hammond, and Rajiv Jain. 2024. Chain of logic: Rule-based reasoning with large language models. *Preprint*, arXiv:2402.10400.

Hongda Sun, Weikai Xu, Wei Liu, Jian Luan, Bin Wang, Shuo Shang, Ji-Rong Wen, and Rui Yan. 2024a. DetermLR: Augmenting LLM-based logical reasoning from indeterminacy to determinacy. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9828–9862, Bangkok, Thailand. Association for Computational Linguistics.

Wangtao Sun, Chenxiang Zhang, XueYou Zhang, Xuanqing Yu, Ziyang Huang, Pei Chen, Haotian Xu, Shizhu He, Jun Zhao, and Kang Liu. 2024b. Beyond instruction following: Evaluating inferential rule following of large language models. *Preprint*, arXiv:2407.08440.

Oyvind Tafjord, Bhavana Dalvi Mishra, and Peter Clark. 2021. Proofwriter: Generating implications, proofs, and abductive statements over natural language. *Preprint*, arXiv:2012.13048.

Jidong Tian, Yitian Li, Wenqing Chen, Liqiang Xiao, Hao He, and Yaohui Jin. 2021. Diagnosing the first-order logical reasoning ability through LogicNLI. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3738–3747, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10014–10037, Toronto, Canada. Association for Computational Linguistics.

Xi Wang, Taketomo Isazawa, Liana Mikaelyan, and James Hensman. 2025. KBLam: Knowledge base augmented language model. In *The Thirteenth International Conference on Learning Representations*.

Fangzhi Xu, Zhiyong Wu, Qiushi Sun, Siyu Ren, Fei Yuan, Shuai Yuan, Qika Lin, Yu Qiao, and Jun Liu. 2024. Symbol-LLM: Towards foundational symbol-centric interface for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13091–13116, Bangkok, Thailand. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Bohan Yu and Yekun Chai. 2025. Evolkv: Evolutionary kv cache compression for llm inference. *Preprint*, arXiv:2509.08315.

Bohan Yu, Wei Huang, and Kang Liu. 2026. Sr-ki: Scalable and real-time knowledge integration into llms via supervised attention. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(41):34486–34494.

Ningyu Zhang, Yunzhi Yao, Jiaxin Qin, Haoming Xu, Yuqi Zhu, Zeping Yu, Mengru Wang, Yuqi Tang, Jia-Chen Gu, Shumin Deng, and Huajun Chen. 2026. Towards principled knowledge editing methods for large language model reasoning. *Nature Machine Intelligence*, 8(8):1189–1200.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, and 3 others. 2026. A survey of large language models. *Preprint*, arXiv:2303.18223.

Ruiwen Zhou, Wenyue Hua, Liangming Pan, Sitao Cheng, Xiaobao Wu, En Yu, and William Yang Wang. 2025. Rulearena: A benchmark for rule-guided reasoning with llms in real-world scenarios. *Preprint*, arXiv:2412.08972.

A Detailed Description of the RuleWorld Benchmark

A.1 Data Statistics

RuleWorld contains 4.94 million abstract, conflict-free procedural rules spanning four major types and seven relation sub-types (Table 5). All rules are fully grounded and self-consistent, allowing direct injection into any QA instance without additional post-processing, which supports systematic evaluation under large-scale rule injection. Here, "A2B" denotes that A influences or determines B, such as *Entity2Attr* and *State2Attr*. Most rules belong to *Entity2Attr* and *State2Attr*, with 1.87M and 3.05M instances, respectively. To build large-scale coverage, we generate many non-commonsense *Entity2Attr* rules as the atomic basis of RuleWorld, while keeping other relation types at moderate scale to maintain balanced reasoning patterns. Together, these relations interact compositionally and support the construction of a large and diverse QA space.

Major Type	Relation Type	All Rules	Train/Eval Rules
Attribute	Entity2Attr	1,877,380	74,888
	AttrChange2Attr	4,021	3,772
Action	Action2Env	100	100
	Action2Attr	200	200
	Action2State	200	200
Environment	Env2State	100	100
State	State2Attr	3,054,112	20,975
Total		**4,936,113**	**100,235**

Table 5: Count distribution of full rule set and the subset used for training and evaluation.

QA Type	Sub-task Type	All QA Count	Train QA Count
Single-Rule QA	single-rule	50,000	25,000
Parallel Multi-Rule QA	multi-rule-2	154,167	6,702
	multi-rule-3	381,116	8,696
	multi-rule-4	839,755	12,384
	multi-rule-5	886,455	10,924
	multi-rule-6	618,577	6,824
	multi-rule-7	289,761	2,863
	multi-rule-8	80,169	807
Multi-Hop Rule QA	multi-hop-2	37,737	19,500
	multi-hop-3	30,000	12,500
	multi-hop-4	10,000	5,000
Total		**3,377,737**	**111,200**

Table 6: Count distributions for full and training QA instances across RuleWorld QA types.

Based on this rule corpus, we instantiate 3.37 million rule-based QA pairs spanning three reasoning settings: Single-Rule QA, Parallel Multi-Rule QA, and Multi-Hop Rule QA (Table 6). Parallel Multi-Rule QA covers up to eight gold rules per instance, while Multi-Hop Rule QA spans up to four reasoning hops, yielding eleven sub-tasks of increasing complexity. Most QA instances come from Parallel Multi-Rule QA, which contributes 3.25M examples and reflects the combinatorial dif-

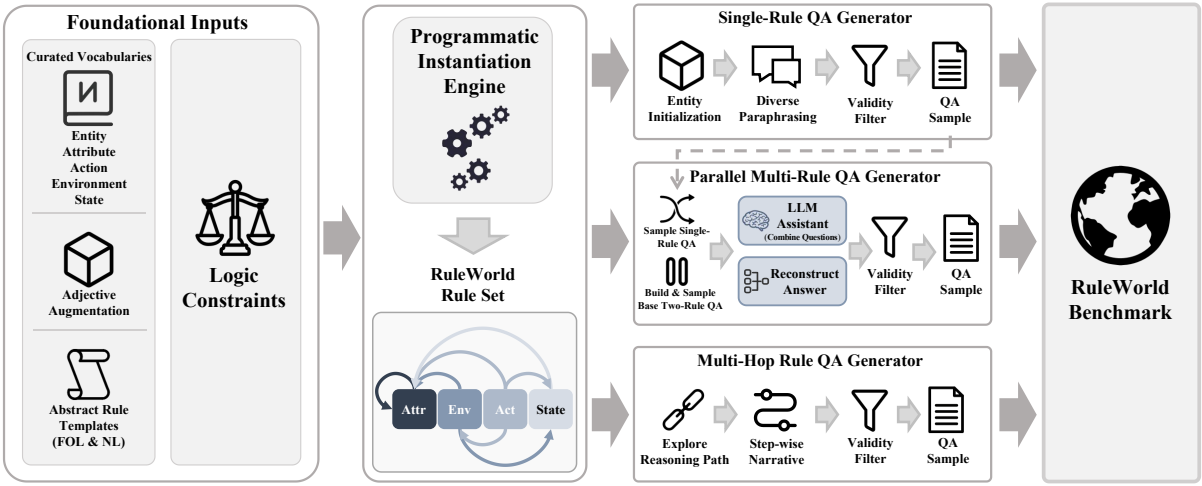


Figure 7: Illustration of the construction pipelines for the RuleWorld rule set (**left**) and the RuleWorld QA benchmark (**right**), including single-rule, parallel multi-rule, and multi-hop QA generation. The modular process integrates curated vocabularies, logic constraints, programmatic instantiation, and task-specific generation workflows. In the rule set, Attr, Env, Act, and State denote the four major types *Attribute*, *Environment*, *Action*, and *State*, and the directed arrows between them indicate their interaction relations.

ficulty of jointly applying multiple procedural rules. In contrast, Multi-Hop QA contains 77,737 instances and focuses on sequential inference across reasoning steps. This distribution supports fine-grained analysis of model behavior under varying reasoning depth and rule composition complexity.

For training and evaluation, we further sample 100,235 rules from the full corpus while preserving its structural distribution (Table 5). Based on these sampled rules, we construct a training subset of 111,200 QA instances (Table 6), including 25,000 Single-Rule, 49,200 Parallel Multi-Rule, and 37,000 Multi-Hop examples. The test split uses the same sampled rules but entirely disjoint QA instances, and is uniformly sampled across difficulty levels with manual verification, ensuring balanced and reliable evaluation.

A.2 Construction Process

As shown in Figure 7, RuleWorld is primarily built through a programmatic generation framework that synthesizes both the rule set and the QA instances at scale. For *Parallel Multi-Rule QA*, LLMs are further used to merge multiple sub-questions into coherent composite questions. To ensure reliable evaluation, we manually annotate the *Parallel Multi-Rule QA* portion of the test sets.

Rule Set Construction As illustrated in Figure 7, RuleWorld begins with predefined vocabularies of *entities*, *attributes*, *actions*, *environments*, and *states*, which define the synthetic world. We further expand lexical diversity by applying adjective-based augmentation, especially to *entities*, *attributes*, and *states*. Based on these vocabularies, we specify abstract interaction patterns, such as action to attribute, state to attribute, or environment to state, and instantiate them into rule templates in both NL and FOL formats. A programmatic engine then enumerates and samples valid condition effect combinations to generate millions of rules. To ensure logical soundness, all generated rules are filtered through consistency checks. In particular, we remove cyclic or alternative causal paths for single-step "A2B" transformations, so that each atomic rule corresponds to a unique and unambiguous mapping. Although individual rules are strictly pruned, the resulting rule set still supports rich multi-hop compositions. This pipeline yields a grounded, conflict-free, and semantically diverse rule set for controlled reasoning evaluation.

Single-Rule QA Construction Given the rule set, we construct Single-Rule QA instances to evaluate whether a model can apply one atomic rule under a controlled initial state. For each rule type, we instantiate diverse question templates to reduce stylistic overfitting. Figures 15–17 show a superset of templates used across the dataset, covering multiple linguistic forms such as conditional, temporal, interrogative, and voice-variant expressions.

For each instance, the system assigns an initial entity configuration, applies one selected rule, and generates a question whose answer requires exactly

Major Type	Relation Type	FOL	NL
Attribute	Entity2Attr	Tiny_cat(A) ⇒ Has(Strong_horn, 4)	If A is a tiny cat, it has 4 strong horns.
	AttrChange2Attr	Lose_Strong_fin(A, 1) ⇒ Drop_Mineral_fur(A, 2)	If A loses 1 strong fin, A will drop 2 mineral furs.
Action	Action2Env	Chase(A, B) ⇒ Enter(B, Bridge)	If A chases B, B will enter bridge.
	Action2Attr	Chase(A, B) ⇒ Drop_Floral_paw(B, 1)	If A chases B, B will drop 1 floral paw.
	Action2Attr	Chase(A, B) ⇒ Get_Stormy_ember(A, 2)	If A chases B, A will get 2 stormy embers.
	Action2State	Scratch(A, B) ⇒ Completely_numb(A)	If A scratches B, A will be completely numb.
	Action2State	Scratch(A, B) ⇒ Deeply_glowing(B)	If A scratches B, B will be deeply glowing.
Environment	Env2State	Desert(A) ⇒ Slightly_disappointed(A)	If A is in desert, A will be slightly disappointed.
State	State2Attr	Deeply_hungry(A) ⇒ Lose_Crystalline_tongue(A, 1)	If A is deeply hungry, A will lose 1 crystalline tongue.

Table 7: Examples of abstract procedural rules represented in FOL and NL formats.

one rule application. Each sample is produced automatically together with structured rule provenance, proof steps, and natural language explanations for supervision and analysis, while evaluation only uses the final answer. Invalid or semantically impossible samples, such as those producing negative attribute values, are removed automatically.

Parallel Multi-Rule QA Construction Building on single-rule QA, we construct Parallel Multi-Rule QA instances by combining multiple independent sub-questions. Here, *parallel* means that the reasoning steps are independent and order-invariant: no step depends on the intermediate result of another, even when multiple steps involve the same entity. As preparation, we first build two-rule QA instances for types such as *AttrChange2Attr*, *Action2Attr*, and *State2Attr*, where an auxiliary *Entity2Attr* rule provides the required initial attribute value. Examples of these templates are shown in Figures 15, 16, and 18.

We then sample from the single-rule and two-rule pools and merge up to four questions into one combined query, covering at most eight atomic rules. The final answer is obtained by executing all rule effects in parallel and aggregating the original sub-question answers. To improve naturalness, we use Qwen2.5-72B-Instruct to merge separate questions into a coherent composite prompt without changing their semantics. The prompt template is shown in Figure 19. Each merged instance inherits the rule metadata of its constituent samples, enabling precise tracking of rule width and structure. Invalid merged samples are filtered using the same criteria as in single-rule QA construction.

Multi-Hop Rule QA Construction For sequential reasoning, we construct Multi-Hop Rule QA instances based on predefined reasoning skeletons, such as *Action2Env* → *Env2state* → *State2Attr* → *AttrChange2Attr*. The generator explores these skeletons by sampling valid rule combinations, simulating stepwise state transitions, and checking that every intermediate step is logically consistent. It then forms a question whose answer depends on the final result of the chain. The generated metadata stores the reasoning depth, intermediate proofs, and full derivation trace. During construction, each rule application is also converted into natural language descriptions, and the same validity checks used in single-rule QA construction are applied to remove invalid samples. This process produces multi-hop QA instances with explicit causal dependencies and rich supervision signals.

Together, these procedures define RuleWorld as a scalable rule-centric benchmark that transforms procedural rules into controlled QA tasks. Through deterministic generation and LLM-assisted composition, RuleWorld supports single-rule, parallel, and multi-hop reasoning, enabling fine-grained evaluation of rule retrieval and rule execution in LLMs.

A.3 Data Examples

Table 7 shows representative examples covering all major rule types, where each rule is expressed in both FOL and NL forms. The FOL version provides a precise symbolic specification of entity attributes, state changes, environmental effects, and action-induced transitions, while the NL version preserves the same semantics in human-readable language. For action-triggered effects, RuleWorld further distinguishes changes applied to the acting entity from those applied to the target entity, enabling asymmetric interactions such as "A scratches B" to produce different outcomes for A and B. These paired forms constitute the atomic rule units underlying all RuleWorld QA tasks.

Figures 22 and 23 present representative exam-

ples of the three QA types: Single-Rule, Parallel Multi-Rule, and Multi-Hop Rule QA. Single-Rule questions require one atomic inference. Parallel Multi-Rule QA combines multiple independent reasoning threads in a single prompt, where each sub-question corresponds to one rule application step. Multi-Hop QA requires sequential reasoning over causally connected rules, where one inferred result triggers the next. Together, these examples illustrate how atomic rules can be composed into diverse reasoning trajectories, creating a large and structured space of QA instances for evaluating rule retrieval and application.

These examples also reflect the training format used in our Stacked Step-Level Attention Training. After each `[Step]` explanation, we insert a special token `<search>` to guide the model toward retrieving the rule needed for the next step. This encourages clear step-level attention separation and supports learning step-level rule application during autoregressive generation.

B Training and Evaluation Settings

Training Settings We freeze the base model and randomly initialize $\tilde{\mathbf{W}}_K^l$ and $\tilde{\mathbf{W}}_V^l$, while copying $\mathbf{W}_Q^l$ to $\tilde{\mathbf{W}}_Q^l$. Training is performed on 4 A800 80GB GPUs using DeepSpeed ZeRO-2 (Rajbhandari et al., 2020) with CPU offloading and bf16 precision. We inject up to 100 rules for confidence layer identification in the first stage and 1000 rules for stacked step-level attention training, using top-100 selection. In the latter stage, we set $\mathcal{T} = 0.05$, add `<search>` to the tokenizer and the model's special-token list, resize the token embedding matrix accordingly, and unfreeze the embedding layer so that the `<search>` embedding can be learned. For end-to-end baselines, we use the model architectures provided in the released official implementation repository and train them with the same training configuration. The per-device batch size is 10 with gradient accumulation 5 (effective batch size 50 per device; global batch size 200 across 4 GPUs), optimized via a cosine scheduler (learning rate 1×10^{-4}, warm-up ratio 1×10^{-2}, weight decay 1×10^{-4}). For training, we construct a RuleWorld subset containing 100K rules and 110K QA instances, as shown in Tables 5 and 6 in Appendix A.1. Each batch consists of 30% Single-Rule, 40% Parallel Multi-Rule, and 30% Multi-Hop instances to ensure balanced reasoning coverage.

Evaluation Settings We use the following evaluation settings:

- **Evaluation data.** All evaluations use five random seeds, with 10 samples per task type across difficulty levels. This yields 110 samples per seed and 550 questions in total. The sampled parallel multi-rule instances are manually checked to ensure correct question combination. We use the same underlying rule pool as in training, but all QA instances come from a held-out set that is not used during training.

- **Rule injection and retrieval.** When the rule set size is 100, we inject all rules. For larger rule set sizes, we apply top-K=100 selection. KBLaM and the prompting baseline inject all rules without retrieval.

- **Baselines.** For RAG baselines, we use BM25 (Robertson et al., 1994) and Qwen3-Embedding-8B as retrievers. End-to-end approaches perform confidence-layer retrieval. For hybrid search, we use Reciprocal Rank Fusion (RRF) (Cormack et al., 2009) with $k = 60$ to fuse BM25 and dense retrieval results.

- **Answer evaluation.** Performance is measured by exact match accuracy on the comma-separated answer list in `\boxed{}`. After normalization, including lowercasing, removing LaTeX or string artifacts, and stripping extra whitespace, an instance scores 0 if the predicted and gold lists differ in length. Otherwise, its score is the fraction of position-wise elements that exactly match the gold answers.

- **Step-wise retrieval evaluation.** For DynaRule's multi-step retrieval, we compute step-wise recall by matching each retrieval step to the corresponding gold step in the reference solution. For example, the first retrieval is evaluated against the Step 1 gold rules. Missing steps are assigned a recall of 0, and extra steps are truncated.

- **Inference setup.** For inference, we set temperature to 0.6 for Qwen3-32B and 0 for other models. The QA template for prompting, which is also used for RAG, is provided in Figure 21.

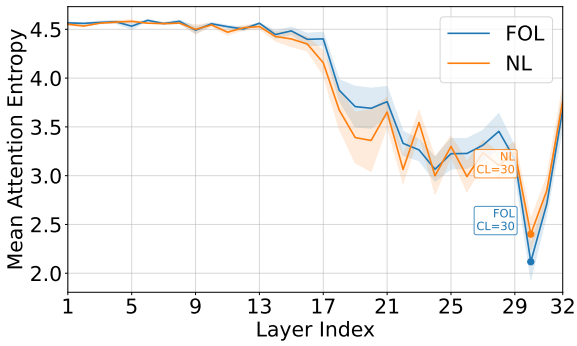
C Extended Experiments and Analysis

C.1 Confidence Layer Identification across Different Model Sizes, Series and Encoders

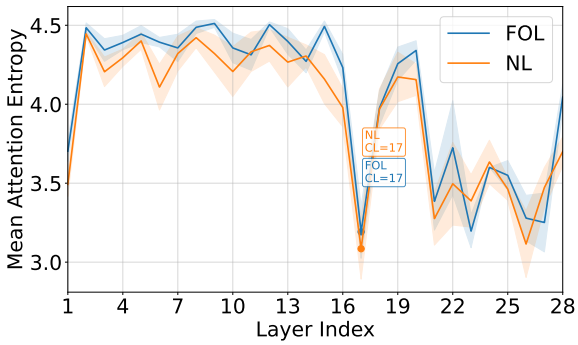
In this experiment, we identify the confidence layer by injecting 100 rules into different base models, including Llama-3-8B-Instruct, Qwen2.5-1.5B-Instruct, and Qwen2.5-3B-Instruct, using Qwen3-Embedding-8B as the encoder. Across both FOL and NL rule formats, as shown in Figures 8a, 8b, 8c, all models consistently exhibit a clear confidence layer, and notably, each model's confidence layer emerges at the same depth for both formats, specifically at layers 30, 17, and 36 respectively. We further evaluate Qwen2.5-7B-Instruct using bge-m3 as the text encoder and observe that the confidence layer again appears at an identical position under FOL and NL rule injections, consistently located at layer 22, as demonstrated in Figure 8d. These results demonstrate that KV-cache-based rule injection reliably yields a stable and format-invariant confidence layer across different model sizes, model series, and encoders, highlighting the robustness of our method.

C.2 Prompting Results with Strong LLMs

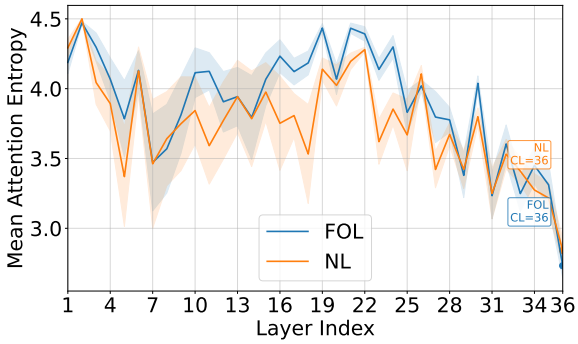
Across both the FOL and NL settings, the prompting results in Table 13 reveal four main findings. **(1)** Existing LLMs struggle to maintain stable reasoning performance as the number of injected rules increases. While several models perform reasonably under 100 rules, their accuracy drops substantially at 500 and 1000 rules, especially on parallel multi-rule reasoning and 3–4 hop multi-hop QA. **(2)** Under most settings, claude-sonnet-4-6 is the strongest prompting baseline overall, particularly on multi-hop reasoning, while Qwen3-32B also demonstrates highly competitive performance under 100-rule setting. As discussed in Appendix C.3, this advantage appears to come from more faithful operator-level rule following and fewer missed rules on long reasoning chains. **(3)** As the rule scale grows to 500 and 1000, most models show pronounced degradation, and Qwen3-32B no longer maintains its small-scale advantage. In these higher-load settings, GPT5.5 and claude-sonnet-4-6 become the strongest baselines, suggesting better robustness to large candidate rule pools. **(4)** Despite the strength of these large prompting baselines, DynaRule consistently achieves the best overall results across all rule scales and both repre-



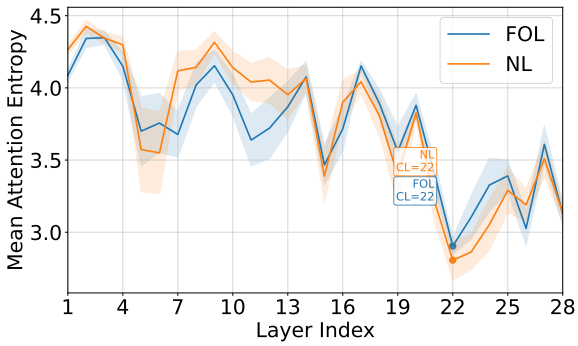
(a) CL identification in Llama-3-8B-Instruct, with layer 30 identified as the CL.



(b) CL identification in Qwen2.5-1.5B-Instruct, with layer 17 identified as the CL.



(c) CL identification in Qwen2.5-3B-Instruct, with layer 36 identified as the CL.



(d) CL identification with bge-m3 as encoder, with layer 22 identified as the CL.

Figure 8: Extended confidence layer (CL) identification under FOL and NL rule representations. Shaded regions denote entropy standard deviation.

sentations. It surpasses the strongest baseline by up to 13.86 points on average and by up to 36 points on the best subtask, demonstrating the effectiveness of step-level dynamic rule integration.

C.3 Case Study

To illustrate how LLMs fail under complex rule interactions, we present three case studies highlighting typical errors: rule omission, incorrect application, and action-type misalignment.

Rule Omission In the first case, as shown in Figure 12, the model is required to determine how many ashen frosts a big fox has when placed in a celestial garden. Although the rule chain clearly specifies a cascading effect: "*Celestial_garden(A)*" induces "*Deeply_starving(A)*", which further triggers "*Drop_Forbidden_void(A, 3)*", and the loss of Forbidden Voids in turn causes "*Drop_Ashen_frost(A, 2)*", the model fails to propagate through this full multi-step sequence. Instead, it only identifies the initial attribute rule "*Big_fox(A)* $\Rightarrow$ *Has(Ashen_frost, 7)*" and entirely omits the subsequent dependency leading to ashen-frost reduction. This omission produces the incorrect final answer of 7, while the correct reasoning yields $1 = (7 - 3 \times 2)$. This demonstrates that even when all rules are available, the model may prematurely terminate the reasoning chain and overlook critical transitions.

Incorrect Use The second case involves computing how many crimson essences a tiny crocodile possesses after being bound by another entity, as illustrated in Figure 13. Here, the model successfully identifies several relevant rules, including the induced state transitions from "*Bind(A, B)*" to "*Enter(B, Starship_deck)*" and subsequently to "*Completely_starving(B)*". However, it misinterprets the rule that governs attribute changes: the model incorrectly applies the rule "*Get_Celestial_fin(A, 1)* $\Rightarrow$ *Gain_Crimson_essence(A, 2)*" assuming that "gaining a celestial fin" automatically counts as "getting" one. This confusion leads the model to treat the growth of 3 celestial fins as the gain of only one fin, resulting in a total of 10 crimson essences, while the correct answer should be $14 = (8+3\times2)$. This reflects a deeper failure in aligning rule semantics, misidentifying rule triggers and misapplying effect operators.

Action-type Misalignment As shown in Figure 14, the model makes an action-type misalignment error in which a neutral action operator such as "gain" fails to trigger when the system encounters a related but distinct operator, "get". Although the rules specify that gaining 1 eternal soul leads to receiving 3 muddy livers, the model incorrectly treats "get 2 eternal souls" as unrelated rather than decomposing it into two unit gain events or recognizing its semantic compatibility. As a result, the inference chain is prematurely terminated: the model concludes that no rule affects muddy livers after the eternal soul change, leading to the incorrect retention of 10 muddy livers. The correct reasoning requires triggering the gain rule twice, producing the correct result of 16 muddy livers. This example demonstrates that when a model rigidly separates action operators without recognizing their functional equivalence, even neutral and intuitively compatible operators fail to activate, causing a silent chain break and ultimately incorrect reasoning outcomes.

Overall, the model frequently misuses injected rules by stopping early, misreading semantics, or missing operator equivalences, resulting in broken reasoning and wrong answers.

C.4 Statistical Robustness

For each method and rule scale, we evaluate five independently sampled, stratified test sets, each containing 110 questions for each rule representation. Thus, each scale involves 1,100 QA evaluations per method across FOL and NL. For QA, we first compute one seed-level overall score by averaging the displayed Single, PM(2–4), PM(5–8), MH-2, MH-3, and MH-4 scores under both FOL and NL. For retrieval, we average Recall@100, Recall@10, and Recall@1 under both representations. We then compute the sample standard deviation over the five seed-level scores. Tables 10 and 11 report the standard deviation and the t-based 95% confidence-interval half-width in percentage points (*std.* / $\pm$CI), where the half-width is $t_{0.975,4} \cdot \mathrm{std}/\sqrt{5}$.

Overall, the seed-level variation is modest and the confidence intervals are much smaller than the main performance gaps at large rule scales. At 10K rules, DynaRule's QA confidence interval is $\pm$4.3 points, while its gains over SR-KI and Hybrid RAG are +19.9 and +37.6 points, respectively. Its retrieval confidence interval is only $\pm$1.3 points.

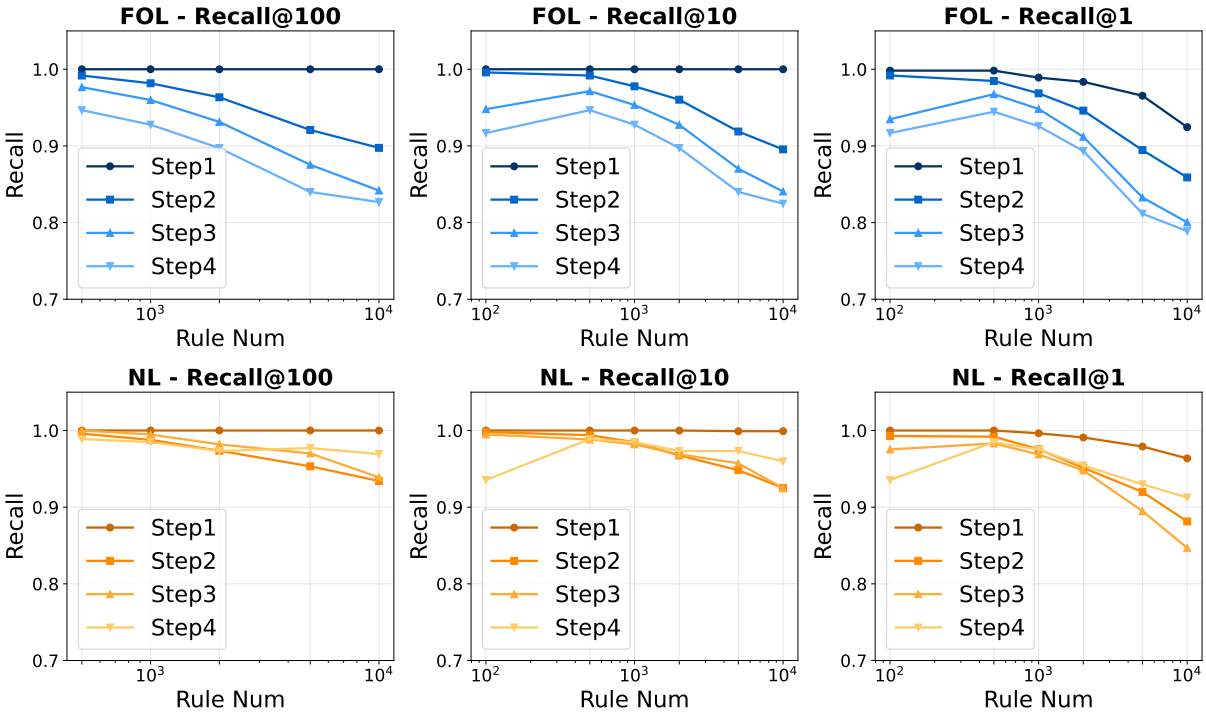


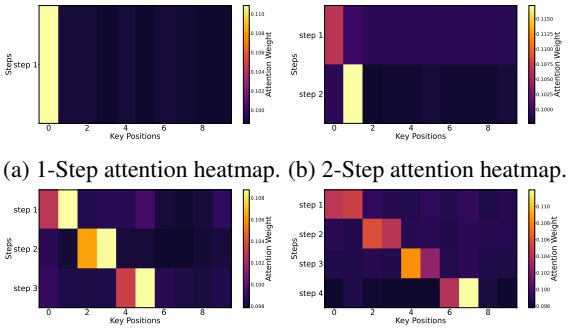
Figure 9: Retrieval performance in terms of Recall@100, Recall@10, and Recall@1 across different reasoning steps and numbers of injected rules under both FOL and NL representations.

Method	Single-Rule QA	Parallel Multi-Rule QA							Multi-Hop Rule QA			Avg.
	single-rule	multi-rule-2	multi-rule-3	multi-rule-4	multi-rule-5	multi-rule-6	multi-rule-7	multi-rule-8	multi-hop-2	multi-hop-3	multi-hop-4	
Rule Num = 100												
Prompting	0.8200	0.7000	0.7367	0.7800	0.6983	0.5183	0.4650	0.2850	0.5000	0.0600	0.0400	0.5094
KBLaM	0.9800	**1.0000**	0.9817	**1.0000**	**1.0000**	0.9833	**0.9750**	0.9900	**1.0000**	0.9000	0.9400	0.9773
SR-KI	**1.0000**	**1.0000**	**0.9950**	0.9700	0.9950	0.9550	**0.9750**	0.9750	0.9800	0.9200	0.9000	0.9695
DynaRule	**1.0000**	**1.0000**	**0.9950**	0.9800	0.9950	**0.9850**	0.9700	**0.9950**	0.9600	**0.9300**	**0.9500**	**0.9782**
Rule Num = 500												
Prompting	0.7400	0.6900	0.6867	0.7583	0.5333	0.4383	0.2700	0.1500	0.3400	0.0400	0.0000	0.4224
RAG_{dense}	0.7800	0.5100	0.4350	0.3467	0.3317	0.2450	0.2200	0.1750	0.0200	0.0200	0.0400	0.2839
RAG_{bm25}	0.6400	0.5300	0.6267	0.5400	0.5717	0.3283	0.3300	0.2950	0.0200	0.0200	0.0200	0.3565
RAG_{hybrid}	0.6800	0.7100	0.6250	0.5217	0.5583	0.4517	0.3400	0.2500	0.0400	0.0200	0.0200	0.3833
KBLaM	0.9600	0.9900	0.9333	0.9550	0.9683	0.9733	0.9500	0.9650	0.9200	0.6200	0.6400	0.8977
SR-KI	0.9800	0.9700	0.9817	0.9550	0.9733	0.9300	0.9250	0.9400	0.8000	0.3400	0.2600	0.8232
DynaRule	**1.0000**	**1.0000**	**0.9867**	**0.9950**	**0.9833**	**1.0000**	**0.9650**	**0.9850**	**0.9800**	**0.7200**	**0.8000**	**0.9468**
Rule Num = 1000												
Prompting	0.8000	0.5900	0.5800	0.6933	0.4533	0.3133	0.2150	0.0800	0.2800	0.0400	0.0000	0.3677
RAG_{dense}	0.6600	0.4400	0.3433	0.2933	0.2467	0.2100	0.2050	0.1500	0.0400	0.0200	0.0200	0.2389
RAG_{bm25}	0.5200	0.6100	0.5600	0.4750	0.4633	0.2717	0.2250	0.1700	0.0000	0.0000	0.0000	0.2995
RAG_{hybrid}	0.6200	0.6300	0.6600	0.5233	0.4467	0.3600	0.3250	0.2400	0.0000	0.0000	0.0200	0.3477
KBLaM	0.9000	0.9500	0.8133	0.8284	0.8867	0.9083	0.8650	0.8950	0.8400	0.4200	0.4400	0.7952
SR-KI	0.9800	0.9600	0.9750	0.9100	0.9433	0.8733	0.8450	0.8500	0.7800	0.2200	0.2800	0.7833
DynaRule	**1.0000**	**1.0000**	**0.9917**	**0.9875**	**0.9792**	**0.9625**	**0.9812**	**0.9688**	**0.8500**	**0.6250**	**0.6500**	**0.9087**
Rule Num = 2000												
Prompting	0.6800	0.4700	0.5267	0.5900	0.4717	0.2500	0.1950	0.0750	0.1000	0.0200	0.0400	0.3108
RAG_{dense}	0.6200	0.2700	0.3167	0.2200	0.2383	0.1783	0.1400	0.0900	0.0000	0.0000	0.0400	0.1921
RAG_{bm25}	0.5600	0.4700	0.4050	0.2850	0.2500	0.1783	0.1400	0.1400	0.0000	0.0000	0.0000	0.2208
RAG_{hybrid}	0.6000	0.6300	0.5633	0.4483	0.3500	0.3083	0.2850	0.1750	0.0200	0.0000	0.0000	0.3073
KBLaM	0.0400	0.0700	0.0133	0.1300	0.0800	0.1100	0.0750	0.0450	0.0200	0.0200	0.0000	0.0548
SR-KI	**1.0000**	0.9300	0.9050	0.8650	0.8583	0.8217	0.7750	0.7750	0.7000	0.1600	0.2000	0.7264
DynaRule	**1.0000**	**0.9800**	**0.9734**	**0.9833**	**0.9400**	**0.9400**	**0.9350**	**0.9350**	**0.8600**	**0.6000**	**0.4600**	**0.8733**
Rule Num = 5000												
RAG_{dense}	0.5800	0.2700	0.2400	0.2167	0.1950	0.1067	0.0800	0.1000	0.0000	0.0000	0.0000	0.1626
RAG_{bm25}	0.5000	0.3800	0.2383	0.1250	0.1167	0.1117	0.1150	0.0750	0.0000	0.0000	0.0000	0.1511
RAG_{hybrid}	0.5800	0.4200	0.4466	0.3533	0.2950	0.2583	0.2600	0.1400	0.0400	0.0200	0.0400	0.2594
KBLaM	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
SR-KI	0.9000	0.8000	0.7817	0.7250	0.6717	0.6550	0.6050	0.5900	0.5800	0.1800	**0.2200**	0.6099
DynaRule	**0.9200**	**0.9400**	**0.9567**	**0.9567**	**0.9167**	**0.9150**	**0.8550**	**0.8650**	**0.6600**	**0.3200**	0.1800	**0.7714**
Rule Num = 10000												
RAG_{dense}	0.4000	0.3000	0.2367	0.1867	0.1617	0.1083	0.0800	0.0850	0.0000	0.0400	0.0000	0.1453
RAG_{bm25}	0.3400	0.2800	0.1667	0.1483	0.1267	0.1183	0.0850	0.0500	0.0200	0.0000	0.0000	0.1214
RAG_{hybrid}	0.4400	0.5000	0.4000	0.3067	0.2650	0.1500	0.1650	0.1400	0.0000	0.0200	0.0000	0.2170
KBLaM	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
SR-KI	0.7400	0.6800	0.6300	0.5483	0.5550	0.5333	0.4600	0.4300	0.3800	**0.2000**	0.1200	0.4797
DynaRule	**0.9200**	**0.9100**	**0.9300**	**0.9233**	**0.8417**	**0.8250**	**0.7550**	**0.7400**	**0.5200**	0.1600	**0.1400**	**0.6968**

Table 8: Exact match accuracy on RuleWorld (**FOL**), using Qwen2.5-7B-Instruct as the base model. RAG_{dense} and RAG_{bm25} use Qwen3-Embedding-8B and BM25 retrievers, respectively; RAG_{hybrid} uses RRF with the fusion constant set to 60 to fuse BM25 and dense retrieval results; suffixes denote the gold parallel rule count or hop number.

Method	Single-Rule QA	Parallel Multi-Rule QA							Multi-Hop Rule QA			Avg.
	single-rule	multi-rule-2	multi-rule-3	multi-rule-4	multi-rule-5	multi-rule-6	multi-rule-7	multi-rule-8	multi-hop-2	multi-hop-3	multi-hop-4	
Rule Num = 100												
Prompting	0.8200	0.8600	0.7833	0.8150	0.7617	0.5217	0.5350	0.4550	0.3400	0.1000	0.1200	0.5556
KBLaM	**1.0000**	**1.0000**	0.9950	0.9800	**0.9950**	**0.9850**	0.9700	**0.9950**	**0.9600**	0.4800	**0.7200**	0.9164
SR-KI	**1.0000**	0.9800	**1.0000**	0.9450	**0.9950**	0.9583	0.9200	0.9600	**0.9600**	0.5400	0.5600	0.8926
DynaRule	**1.0000**	**1.0000**	0.9950	**0.9950**	0.9900	0.9800	**0.9750**	0.9850	0.9200	**0.9600**	0.7000	**0.9545**
Rule Num = 500												
Prompting	0.7400	0.7500	0.7367	0.7217	0.5550	0.3733	0.2700	0.1950	0.1600	0.0400	0.0200	0.4147
RAG_{dense}	0.8600	0.6300	0.6150	0.4716	0.4983	0.2483	0.3450	0.3000	0.0600	0.0400	0.0000	0.3698
RAG_{bm25}	0.7600	0.4500	0.3000	0.3183	0.4217	0.2433	0.2100	0.1750	0.0400	0.0400	0.0200	0.2708
RAG_{hybrid}	0.8400	0.7200	0.5850	0.4967	0.4983	0.3717	0.3900	0.3300	0.1400	0.0000	0.0200	0.3992
KBLaM	0.9600	0.9700	0.9183	0.9133	0.9483	0.9600	**0.9650**	**0.9750**	0.8200	0.5600	0.7600	0.8864
SR-KI	**1.0000**	0.9500	0.9300	0.9067	0.9600	0.8983	0.8700	0.8800	0.6600	0.2600	0.1200	0.7668
DynaRule	0.9800	**0.9800**	**0.9950**	**1.0000**	**0.9783**	**0.9800**	0.9600	**0.9750**	**0.8800**	**0.7000**	**0.8200**	**0.9317**
Rule Num = 1000												
Prompting	0.6200	0.6300	0.6500	0.6617	0.5333	0.2833	0.2200	0.0800	0.0600	0.0200	0.0000	0.3417
RAG_{dense}	0.8800	0.6900	0.6100	0.4250	0.3333	0.1800	0.2000	0.2400	0.0200	0.0000	0.0000	0.3253
RAG_{bm25}	0.5800	0.4100	0.3400	0.3817	0.3950	0.2617	0.1900	0.1050	0.0000	0.0000	0.0000	0.2421
RAG_{hybrid}	0.8200	0.5800	0.5133	0.4967	0.3667	0.2650	0.2800	0.3500	0.1600	0.0000	0.0200	0.3502
KBLaM	0.7800	0.8100	0.8383	0.8333	0.8683	0.8833	0.8700	0.8700	0.6800	0.3400	0.4200	0.7448
SR-KI	**0.9800**	0.8700	0.9033	0.8567	0.9133	0.8417	0.7700	0.7800	0.5800	0.1400	0.0400	0.6977
DynaRule	**0.9800**	**0.9800**	**0.9817**	**0.9883**	**0.9783**	**0.9783**	**0.9550**	**0.9450**	**0.8400**	**0.6400**	**0.6800**	**0.9042**
Rule Num = 2000												
Prompting	0.5600	0.5500	0.6833	0.5950	0.3600	0.3050	0.1700	0.0700	0.0400	0.0400	0.0200	0.3085
RAG_{dense}	0.7600	0.5700	0.4500	0.4117	0.2517	0.1233	0.1700	0.1500	0.0400	0.0000	0.0400	0.2697
RAG_{bm25}	0.7000	0.3700	0.2700	0.3433	0.3667	0.1867	0.1450	0.0800	0.0400	0.0200	0.0200	0.2311
RAG_{hybrid}	0.8200	0.5100	0.5200	0.3967	0.4350	0.3067	0.2300	0.1900	0.1200	0.0600	0.0400	0.3299
KBLaM	0.4400	0.5800	0.5200	0.5217	0.4783	0.4900	0.4950	0.4300	0.3400	0.2000	0.1200	0.4195
SR-KI	0.9000	0.8700	0.7983	0.7583	0.7900	0.7567	0.6350	0.6900	0.5000	0.1200	0.1400	0.6326
DynaRule	**0.9800**	**0.9000**	**0.9583**	**0.9667**	**0.9317**	**0.9517**	**0.9250**	**0.8600**	**0.7000**	**0.3600**	**0.5200**	**0.8230**
Rule Num = 5000												
RAG_{dense}	0.7200	0.3800	0.3617	0.2600	0.2200	0.1150	0.1150	0.0650	0.0200	0.0200	0.0600	0.2124
RAG_{bm25}	0.5800	0.2900	0.2900	0.3183	0.2950	0.1767	0.1200	0.0650	0.0400	0.0000	0.0200	0.1995
RAG_{hybrid}	0.7000	0.4500	0.4133	0.3950	0.4633	0.2400	0.2250	0.1350	0.0400	0.0000	0.0000	0.2783
KBLaM	0.0000	0.0300	0.0200	0.0067	0.0050	0.0100	0.0000	0.0000	0.0000	0.0200	0.0000	0.0083
SR-KI	0.8600	0.7300	0.6367	0.5450	0.5667	0.5267	0.5000	0.5150	0.2200	0.0600	0.0800	0.4764
DynaRule	**0.9400**	**0.8900**	**0.9050**	**0.9167**	**0.8733**	**0.8867**	**0.8300**	**0.7700**	**0.5800**	**0.1800**	**0.3800**	**0.7411**
Rule Num = 10000												
RAG_{dense}	0.6600	0.3200	0.3500	0.1533	0.1667	0.0500	0.0800	0.0400	0.0000	0.0000	0.1000	0.1745
RAG_{bm25}	0.6400	0.2600	0.2600	0.3250	0.3100	0.1550	0.1200	0.0700	0.0000	0.0000	0.0400	0.1982
RAG_{hybrid}	0.6000	0.4200	0.4033	0.3983	0.3600	0.1933	0.1550	0.1550	0.0200	0.0000	0.0400	0.2495
KBLaM	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
SR-KI	0.7400	0.6100	0.5083	0.4217	0.4733	0.4150	0.3600	0.3800	0.1200	0.0800	0.0400	0.3771
DynaRule	**0.8000**	**0.8600**	**0.8550**	**0.8650**	**0.7983**	**0.7767**	**0.6750**	**0.6650**	**0.6000**	**0.1600**	**0.2400**	**0.6632**

Table 9: Exact match accuracy on RuleWorld (**NL**), using Qwen2.5-7B-Instruct as the base model. RAG_{dense} and RAG_{bm25} use Qwen3-Embedding-8B and BM25 retrievers, respectively; RAG_{hybrid} uses RRF with the fusion constant set to 60 to fuse BM25 and dense retrieval results; suffixes denote the gold parallel rule count or hop number.



(a) 1-Step attention heatmap. (b) 2-Step attention heatmap.

(c) 3-Step attention heatmap. (d) 4-Step attention heatmap.

Figure 10: Heatmap of attention weights across reasoning steps. Required rules are ordered for clarity.

C.5 Detailed Retrieval Results across Reasoning Steps and QA Types

We summarize the retrieval results into three main findings. **(1)** Figure 9 shows that step-level retrieval remains strong across both FOL and NL rule representations, with recall generally staying above 0.8 as the rule set size increases from 100 to 10000. In particular, Step 1 remains nearly perfect under all metrics, indicating that early-stage retrieval is highly reliable. **(2)** As reasoning depth increases from Step 2 to Step 4 in Figure 9, retrieval performance gradually declines, especially under stricter metrics such as Recall@1. This trend suggests that deeper multi-step reasoning introduces cumulative retrieval difficulty, with FOL showing a slightly sharper drop than NL at larger rule scales. **(3)** Figure 11 further shows that DynaRule consistently outperforms all baselines across QA types and rule injection settings. Its performance is near perfect on Single-Rule and Parallel Multi-Rule QA, while Multi-Hop Rule QA shows more noticeable degradation, highlighting the greater sensitivity of sequential reasoning to intermediate retrieval errors.

C.6 Attention Weights across Reasoning Steps

We visualize the attention distribution over rule keys under different reasoning depths, with step-specific rules arranged sequentially for clarity, as shown in Figure 10. For single-step tasks, the model concentrates almost entirely on the first re-

Method	100	500	1K	2K	5K	10K
Prompting	1.6 / ±1.9	2.4 / ±3.0	2.0 / ±2.5	4.1 / ±5.1	–	–
$\text{RAG}_{\text{dense}}$	3.1 / ±3.8	3.6 / ±4.5	1.5 / ±1.9	1.7 / ±2.1	2.3 / ±2.9	0.8 / ±1.0
RAG_{bm25}	3.1 / ±3.8	1.1 / ±1.4	1.0 / ±1.2	1.5 / ±1.9	2.7 / ±3.4	3.2 / ±4.0
$\text{RAG}_{\text{hybrid}}$	2.0 / ±2.5	1.1 / ±1.4	2.3 / ±2.9	1.4 / ±1.7	3.5 / ±4.3	2.3 / ±2.9
KBLaM	2.1 / ±2.6	1.8 / ±2.2	2.9 / ±3.6	3.6 / ±4.4	0.7 / ±0.9	0.0 / ±0.0
SR-KI	2.0 / ±2.5	1.7 / ±2.2	0.6 / ±0.8	1.2 / ±1.4	1.8 / ±2.3	3.0 / ±3.7
DynaRule	1.1 / ±1.4	1.7 / ±2.2	3.0 / ±3.7	2.4 / ±3.0	2.6 / ±3.2	3.4 / ±4.3

Table 10: Seed-level variation for exact match accuracy, reported as standard deviation / t-based 95% confidence-interval half-width (percentage points).

Method	100	500	1K	2K	5K	10K
Dense	1.0 / ±1.2	1.7 / ±2.1	1.6 / ±2.0	1.6 / ±1.9	1.5 / ±1.8	1.3 / ±1.6
BM25	0.8 / ±1.1	1.1 / ±1.4	1.2 / ±1.5	1.8 / ±2.2	1.5 / ±1.9	2.0 / ±2.5
Hybrid	0.7 / ±0.8	1.0 / ±1.3	0.6 / ±0.8	0.6 / ±0.7	1.0 / ±1.2	1.4 / ±1.7
KBLaM	1.1 / ±1.3	1.3 / ±1.6	1.3 / ±1.6	1.4 / ±1.7	1.3 / ±1.6	4.1 / ±5.1
SR-KI	0.5 / ±0.6	0.7 / ±0.8	0.6 / ±0.8	1.0 / ±1.3	1.4 / ±1.8	1.3 / ±1.6
DynaRule	0.4 / ±0.4	0.3 / ±0.4	0.6 / ±0.7	0.8 / ±1.0	1.3 / ±1.6	1.1 / ±1.3

Table 11: Seed-level variation for retrieval, reported as standard deviation / t-based 95% confidence-interval half-width (percentage points).

Method	**First-order Logic**			**Natural Language**		
	Recall@100	Recall@10	Recall@1	Recall@100	Recall@10	Recall@1
Rule Num = 100						
Dense	-	0.5325	0.4448	-	0.6646	0.5414
BM25	-	0.6901	0.4375	-	0.5869	0.5292
Hybrid	-	0.6364	0.4793	-	0.5972	0.5128
KBLaM	-	0.5151	0.4122	-	0.5278	0.4025
SR-KI	-	0.9100	0.8000	-	0.8963	0.7768
DynaRule	-	**0.9773**	**0.9733**	-	**0.9909**	**0.9857**
Rule Num = 500						
Dense	0.6208	0.4079	0.3371	0.7700	0.5114	0.4039
BM25	0.8186	0.3934	0.2395	0.6247	0.5136	0.4680
Hybrid	0.8399	0.4376	0.2974	0.8026	0.4965	0.4111
KBLaM	0.6148	0.4137	0.2647	0.6215	0.4044	0.2452
SR-KI	0.9520	0.8047	0.6372	0.9493	0.7886	0.6163
DynaRule	**0.9862**	**0.9852**	**0.9814**	**0.9973**	**0.9941**	**0.9920**
Rule Num = 1000						
Dense	0.5529	0.3635	0.3028	0.6961	0.4490	0.3587
BM25	0.7559	0.2962	0.1887	0.5946	0.4892	0.4424
Hybrid	0.7935	0.3561	0.2456	0.7618	0.4464	0.3839
KBLaM	0.5237	0.3532	0.1739	0.5389	0.3351	0.1686
SR-KI	0.9331	0.7369	0.5318	0.9288	0.7213	0.5429
DynaRule	**0.9762**	**0.9737**	**0.9656**	**0.9932**	**0.9892**	**0.9810**
Rule Num = 2000						
Dense	0.4929	0.3192	0.2717	0.6293	0.3969	0.3150
BM25	0.5916	0.2273	0.1496	0.5792	0.4614	0.4182
Hybrid	0.7167	0.2943	0.1929	0.7118	0.4086	0.3620
KBLaM	0.4728	0.2532	0.1106	0.4734	0.2352	0.1068
SR-KI	0.9070	0.6503	0.4343	0.8989	0.6465	0.4661
DynaRule	**0.9618**	**0.9595**	**0.9464**	**0.9848**	**0.9798**	**0.9640**
Rule Num = 5000						
Dense	0.4232	0.2789	0.2403	0.5282	0.3343	0.2575
BM25	0.4209	0.1548	0.0800	0.5258	0.4299	0.3869
Hybrid	0.5754	0.2570	0.1490	0.6370	0.3806	0.3226
KBLaM	0.4231	0.1297	0.0559	0.4189	0.1289	0.0508
SR-KI	0.8491	0.5168	0.3145	0.8261	0.5402	0.3662
DynaRule	**0.9307**	**0.9286**	**0.8964**	**0.9745**	**0.9682**	**0.9346**
Rule Num = 10000						
Dense	0.3710	0.2573	0.2227	0.4619	0.2940	0.2241
BM25	0.3017	0.1102	0.0418	0.4970	0.4058	0.3703
Hybrid	0.5023	0.2344	0.1338	0.5876	0.3559	0.3023
KBLaM	0.3884	0.0642	0.0239	0.3554	0.0776	0.0282
SR-KI	0.7758	0.4180	0.2415	0.7539	0.4465	0.2898
DynaRule	**0.9145**	**0.9130**	**0.8613**	**0.9614**	**0.9526**	**0.9045**

Table 12: Retrieval performance on FOL and NL rules. Dense uses Qwen3-Embedding-8B; Hybrid uses RRF (fusion constant 60) to combine BM25 and dense retrieval; KBLaM, SR-KI, and DynaRule use confidence-layer retrieval. For Recall@1, K is set to the number of gold rules per instance or step.

quired rule. As the reasoning depth increases, attention shifts progressively toward the rules needed at each step. When a step requires multiple rules, the model jointly attends to them, forming a distinct cluster clearly separated from those used in earlier steps. Overall, this produces a stepwise, staircase-like attention pattern, highlighting the model's ability to dynamically select and apply the correct rules throughout multi-step reasoning.

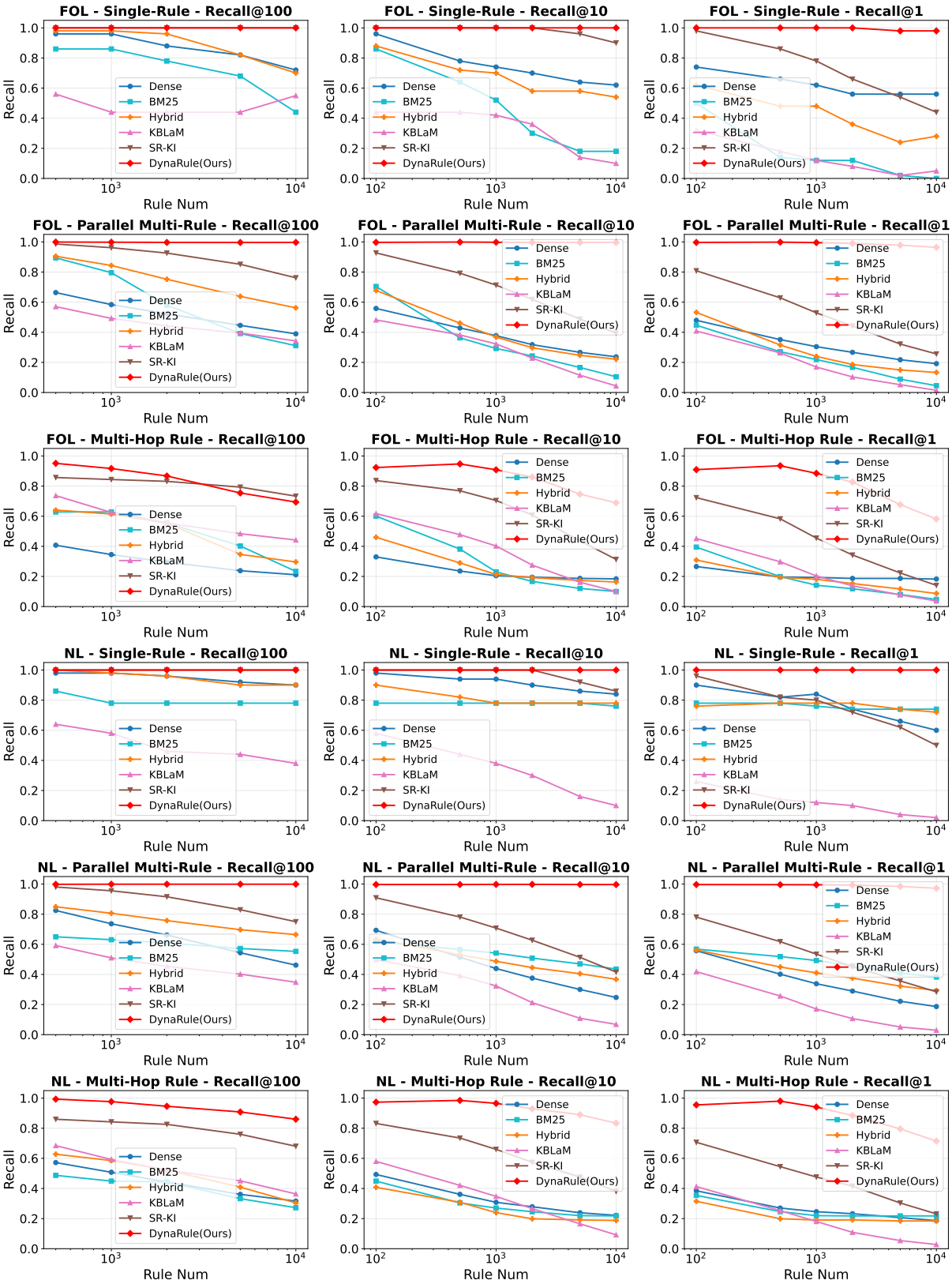


Figure 11: Comparison of retrieval performance across QA types under FOL (top) and NL (bottom) rule representations. Dense uses Qwen3-Embedding-8B; Hybrid uses RRF with a fusion constant of 60; KBLaM, SR-KI, and DynaRule use confidence-layer retrieval. Recall@1 sets K to the number of gold rules per instance or step.

Case Study (Omission)

Gold Rules:
Celestial_garden(A) => Deeply_starving(A)
Deeply_starving(A) => Drop_Forbidden_void(A, 3)
Big_fox(A) => Has(Ashen_frost, 7)
Lose_Forbidden_void(A, 1) => Drop_Ashen_frost(A, 2) **Miss**

Question:
How many ashen frosts will the big fox have if the big fox is located in celestial garden?

Model Outputs:
To determine how many Ashen Frosts a Big Fox will have if it is located in a Celestial Garden, we need to follow the rules step by step:

1. **Identify the initial state of the Big Fox:**
   - According to the rule library, a Big Fox has Ashen Frost:
     \[
     \text{Big_fox(A)} \implies \text{Has(Ashen_frost, 7)}
     \]
   - So, initially, the Big Fox has 7 Ashen Frosts.

2. **Determine the effect of being in the Celestial Garden:**
   - The rule for being in a Celestial Garden is:
     \[
     \text{Celestial_garden(A)} \implies \text{Deeply_starving(A)}
     \]
   - This means that if the Big Fox is in the Celestial Garden, it becomes deeply starving.

3. **Determine the effect of being deeply starving:**
   - The rule for being deeply starving is:
     \[
     \text{Deeply_starving(A)} \implies \text{Drop_Forbidden_void(A, 3)}
     \]
   - However, this rule does not directly affect the number of Ashen Frosts. It only affects Forbidden Voids.

4. **Check for any other rules that might affect Ashen Frosts:**
   - There are no other rules in the library that directly affect the number of Ashen Frosts when a creature is deeply starving or in a Celestial Garden.

Since the only rule affecting the Big Fox in the Celestial Garden does not change the number of Ashen Frosts, the Big Fox will still have 7 Ashen Frosts.

Therefore, the final answer is:
\[
\boxed{7} **Correct: 7 - 3*2 = 1**
\]

Figure 12: Case example of rule omission (missing rule application).

Case Study (Incorrect Use)

Gold Rules:
Bind(A, B) => Enter(B, Starship_deck)
Starship_deck(A) => Completely_starving(A)
Completely_starving(A) => Gain_Celestial_fin(A, 3)
Tiny_crocodile(A) => Has(Crimson_essence, 8)
Get_Celestial_fin(A, 1) => Gain_Crimson_essence(A, 2) **Incorrect Use**

Question:
How many crimson essences does the tiny crocodile have after a medium monkey binds the tiny crocodile?

Model Outputs:
To solve this problem, we need to follow the rules in the RULE LIBRARY step by step to determine how many crimson essences the tiny crocodile has after a medium monkey binds it.

1. **Identify the initial state of the tiny crocodile:**
   - According to the RULE LIBRARY, a `Tiny_crocodile(A)` has `Crimson_essence(A, 8)`.

2. **Understand the binding action:**
   - The rule `Bind(A, B) => Enter(B, Starship_deck)` means that when a medium monkey binds a tiny crocodile, the tiny crocodile enters the starship deck.

3. **Determine the effects of entering the starship deck:**
   - The rule `Starship_deck(A) => Completely_starving(A)` indicates that entering the starship deck causes the tiny crocodile to become completely starving.
   - The rule `Completely_starving(A) => Gain_Celestial_fin(A, 3)` means that being completely starving results in gaining a celestial fin with a value of 3.

4. **Check if gaining a celestial fin affects the number of crimson essences:**
   - The rule `Get_Celestial_fin(A, 1) => Gain_Crimson_essence(A, 2)` indicates that gaining a celestial fin results in gaining 2 more crimson essences.

5. **Calculate the total number of crimson essences:**
   - Initially, the tiny crocodile has 8 crimson essences.
   - After gaining a celestial fin, it gains 2 more crimson essences.
   - Therefore, the total number of crimson essences is \(8 + 2 = 10 \).

Thus, the final number of crimson essences the tiny crocodile has after a medium monkey binds it is \boxed{10}. **Correct: 8 + 3*2 = 14**

Figure 13: Case example of incorrect rule application.

Case Study (Action-type Misalignment)

Gold Rules:
If A believes B, B will enter royal court.
If A is in royal court, A will be severely depressed.
If A is severely depressed, A will get 2 eternal souls.
If A is a medium duck, it has 10 muddy livers.
If A gains 1 eternal soul, A will get 3 muddy livers. **Misalignment**

Question:
How many muddy livers does the medium duck have after the medium duck is believed by a huge cat?

Model Outputs:
1. From the rules:
   - "If A is a medium duck, it has **10 muddy livers**."
   So initially, the medium duck has 10 muddy livers.

2. Another rule:
   - "If A believes B, B will enter royal court."
   We are told "a medium duck is believed by a huge cat," i.e., the huge cat believes the duck.
   So: the huge cat (A) believes the medium duck (B) ⇒ the duck **enters royal court**.

3. Another rule:
   - "If A is in royal court, A will be severely depressed."
   Therefore, the medium duck becomes **severely depressed**.

4. Another rule:
   - "If A is severely depressed, A will get 2 eternal souls."
   This changes eternal souls, but does **not** mention muddy livers.

5. There is also:
   - "If A gains 1 eternal soul, A will get 3 muddy livers."
   However, the duck "gets 2 eternal souls," not "gains 2 eternal soul," and the rules are applied strictly by wording. So this rule does **not** trigger.

No rule changes the number of muddy livers for becoming severely depressed or for getting 2 eternal souls.

Thus, the medium duck still has **10 muddy livers**.

\boxed{10} **Correct: 10 + 2*3 = 16**

Figure 14: Case example of action-type misalignment.

Question Template (Attribute)

Entity2Attr:
How many {attr} does {entity} have?
How many {attr} can be found on {entity}?
How many {attr} does {entity} own?
What is the count of {attr} for {entity}?
Do you know how many {attr} {entity} has?
What total of {attr} does {entity} have?
How many {attr} does {entity} possess?
How many {attr} are possessed by {entity}?
How many {attr} belong to {entity}?
How many {attr} are there on {entity}?

AttrChange2Attr:
If {entity} {action} {delta} {attr1}, how many {attr2} does it have?
If {entity} {action} {delta} {attr1}, what number of {attr2} does it have?
If {entity} {action} {delta} {attr1}, how many {attr2} will it have?
After {entity} {action} {delta} {attr1}, how many {attr2} will it possess?
When {entity} {action} {delta} {attr1}, how many {attr2} will it remain?
If {delta} {attr1} {predicate} {action} by {entity}, how many {attr2} does it have?
If {delta} {attr1} {predicate} {action} by {entity}, how many {attr2} will it have?
If {delta} {attr1} {predicate} {action} by {entity}, how many {attr2} does it possess?
How many {attr2} remain after {entity} {action} {delta} {attr1}?
How many {attr2} does {entity} have once it {action} {delta} {attr1}?
How many {attr2} are left once {entity} {action} {delta} {attr1}?

Figure 15: Basic question templates for Attribute-type rules.

Question Template (Action)

Action2State (Querying the resulting state of A):
If {A} {action} {B}, what state will the {A} be in?
If {A} {action} {B}, what will the {A}'s state be?
If {A} {action} {B}, what is the resulting state of the {A}?
When {A} {action} {B}, what will {A}'s state be?
After {A} {action} {B}, what state will the {A} be in?
If {B} is {action} by {A}, what state will the {A} be in?
If {B} is {action} by {A}, what will the {A}'s state be?
If {B} gets {action} by {A}, what state will the {A} be in?
When {B} is {action} by {A}, what will the {A}'s state be?
After {B} is {action} by {A}, what state will the {A} be in?
What state will the {A} be in if {A} {action} {B}?
What state will the {A} be in if {B} is {action} by {A}?

Action2State (Querying the resulting state of B):
Same templates as above by swapping A and B.

Action2Attr (Querying the resulting state of A):
If {A} {action} {B}, how many {attr} does the {A} have?
If {A} {action} {B}, what number of {attr} does the {A} have?
If {A} {action} {B}, how many {attr} will the {A} have?
When {A} {action} {B}, how many {attr} does the {A} have?
After {A} {action} {B}, how many {attr} does the {A} have left?
If {B} is {action} by {A}, how many {attr} does the {A} have?
When {B} is {action} by {A}, how many {attr} does the {A} have?
After {B} is {action} by {A}, how many {attr} will the {A} have?
How many {attr} does {A} have after it {action} {B}?
How many {attr} will {A} have once it {action} {B}?
How many {attr} does the {A} have once {B} is {action} by {A}?
How many {attr} will the {A} have if {B} is {action} by {A}?

Action2Attr (Querying the resulting state of A):
Same templates as above by swapping A and B.

Action2Env:
If {A} {action} {B}, where the {B} will move to?
If {A} {action} {B}, where will the {B} go?
If {A} {action} {B}, where will the {B} move?
If {A} {action} {B}, what place will the {B} move to?
If {A} {action} {B}, what location will the {B} go to?
If {B} is {action} by {A}, where will the {B} move to?
If {B} is {action} by {A}, where will the {B} go?
If {B} is {action} by {A}, where does the {B} go?
If {B} is {action} by {A}, what place will the {B} go to?
If {B} is {action} by {A}, what location will the {B} go to?
Where will the {B} go if {A} {action} {B}?
Where will the {B} go if {B} is {action} by {A}?
Where does the {B} move to when {A} {action} {B}?
Where does the {B} move to when {B} is {action} by {A}?

Figure 16: Basic question templates for Action-type rules.

Question Template (Environment)

Env2State:
If {entity} is in {env}, what state will the {entity} be in?
If {entity} is in {env}, what will its state be?
If {entity} stays in {env}, what state will it be in?
If {entity} enters {env}, what will its state become?
If {entity} appears in {env}, what will its state become?
When {entity} is in {env}, what state will it be in?
If {entity} finds itself in {env}, what state will it be in?
What state will {entity} be in if it is in {env}?
What state will {entity} be in when it is in {env}?
If {entity} is located in {env}, what will its state be?

Figure 17: Basic question templates for Environment-type rules.

Question Template (State)

State2Attr:
If {entity} is {state}, how many {attr} does it have?
If {entity} becomes {state}, how many {attr} does it have?
If {entity} turns {state}, how many {attr} will it have?
When {entity} is {state}, how many {attr} does it have?
After {entity} becomes {state}, how many {attr} does it have?
How many {attr} will {entity} have if it is {state}?
How many {attr} does {entity} have when it is {state}?
How many {attr} would {entity} have in the {state} state?
When {entity} reaches the {state} state, how many {attr} will it possess?
What number of {attr} does {entity} have when it's {state}?

Figure 18: Basic question templates for State-type rules.

Prompt Template (Combine Questions)

{questions}

Combine these {qa_num} questions into 1 sentence. Do not miss or add any information.

Figure 19: Template for combining questions to form Parallel Multi-Rule QA.

Prompt Template (Train)

Question Template:
Answer the following question based on the RULE LIBRARY:
[QUESTION]
{question}
Let's think step by step and put your final answer within \boxed{}.

Answer Template:
Based on the rules provided:
{proofs}
Therefore, the final answer is \boxed{{answer}}.

Figure 20: Question-answering template for DynaRule training.

Prompt Template (Prompting)

Question Template:
Answer the following question based on the RULE LIBRARY:
[RULE]
{rules}
[QUESTION]
{question}
Let's think step by step and put your final answer within a single \boxed{} like \boxed{1, 3, happy}.

Figure 21: Question-answering template for prompting evaluation.

Samples (FOL)

Single-Rule:
Answer the following question based on the RULE LIBRARY:
[QUESTION]
How many mystic flames can be found on a huge goat?
Let's think step by step and put your final answer within \boxed{}.

Based on the rules provided:
[Step 1]
Rule: Huge_goat(A) => Has(Mystic_flame, 1)
We can infer that a huge goat has 1 mystic flame.
Therefore, the final answer is \boxed{1}.

Parallel Multi-Rule:
Answer the following question based on the RULE LIBRARY:
[QUESTION]
After a medium owl becomes completely satisfied, how many warm teeth does it have, if a big pigeon turns severely excited, how many rough cores will it have, how many luminous mouths does a huge fairy have when it is completely buried, and how many broken scales would a medium griffin have in the severely transformed state?
Let's think step by step and put your final answer within \boxed{}.

Based on the rules provided:
[Step 1]
Rule: Medium_owl(A) => Has(Warm_tooth, 9)
Completely_satisfied(A) => Lose_Warm_tooth(A, 3)
Since A is a medium owl, a medium owl has 9 warm teeth and is completely satisfied, we can infer that the medium owl has (9 - 3) = 6 warm teeth.
[Step 2]
Rule: Big_pigeon(A) => Has(Rough_core, 1) Severely_excited(A) => Gain_Rough_core(A, 2)
Since A is a big pigeon, a big pigeon has 1 rough core and is severely excited, we can infer that the big pigeon has (1 + 2) = 3 rough cores.
[Step 3]
Rule: Huge_fairy(A) => Has(Luminous_mouth, 10)
Completely_buried(A) => Drop_Luminous_mouth(A, 3)
Since A is a huge fairy, a huge fairy has 10 luminous mouths and is completely buried, we can infer that the huge fairy has (10 - 3) = 7 luminous mouths.
[Step 4]
Rule: Medium_griffin(A) => Has(Broken_scale, 7)
Severely_transformed(A) => Drop_Broken_scale(A, 2)
Since A is a medium griffin, a medium griffin has 7 broken scales and is severely transformed, we can infer that the medium griffin has (7 - 2) = 5 broken scales.
Therefore, the final answer is \boxed{6, 3, 7, 5}.

Multi-Hop Rule:
Answer the following question based on the RULE LIBRARY:
[QUESTION]
How many muddy livers does the medium duck have after the medium duck is believed by a huge cat?
Let's think step by step and put your final answer within \boxed{}.

Based on the rules provided:
[Step 1]
Rule: Believe(A, B) => Enter(B, Royal_court)
Since A is a huge cat, B is a medium duck, the huge cat believes the medium duck, we can infer that the medium duck will be in royal court.
[Step 2]
Rule: Royal_court(A) => Severely_depressed(A)
Since A is a medium duck and is in royal court, we can infer that it is severely depressed.
[Step 3]
Rule: Severely_depressed(A) => Get_Eternal_soul(A, 2)
Since A is a medium duck and is severely depressed, we can infer that the medium duck will get 2 eternal souls.
[Step 4]
Rule: Medium_duck(A) => Has(Muddy_liver, 10)
Gain_Eternal_soul(A, 1) => Get_Muddy_liver(A, 3)
Since A is a medium duck, a medium duck has 10 muddy livers, the medium duck gets 2 eternal souls, we can infer that the medium duck has (10 + 2 * 3) = 16 muddy livers.
Therefore, the final answer is \boxed{16}.

Figure 22: Examples of Single-Rule, Parallel Multi-Rule, and Multi-Hop Rule QA under FOL representation.

Samples (NL)

Single-Rule:
Answer the following question based on the RULE LIBRARY:
[QUESTION]
How many mystic flames can be found on a huge goat?
Let's think step by step and put your final answer within \boxed{}.

Based on the rules provided:
[Step 1]
Rule: If A is a huge goat, it has 1 mystic flame.
We can infer that a huge goat has 1 mystic flame.
Therefore, the final answer is \boxed{1}.

Parallel Multi-Rule:
Answer the following question based on the RULE LIBRARY:
[QUESTION]
After a medium owl becomes completely satisfied, how many warm teeth does it have, if a big pigeon turns severely excited, how many rough cores will it have, how many luminous mouths does a huge fairy have when it is completely buried, and how many broken scales would a medium griffin have in the severely transformed state?
Let's think step by step and put your final answer within \boxed{}.

Based on the rules provided:
[Step 1]
Rule: If A is a medium owl, it has 9 warm teeth. If A is completely satisfied, A will lose 3 warm teeth.
Since A is a medium owl, a medium owl has 9 warm teeth and is completely satisfied, we can infer that the medium owl has (9 - 3) = 6 warm teeth.
[Step 2]
Rule: If A is a big pigeon, it has 1 rough core. If A is severely excited, A will gain 2 rough cores.
Since A is a big pigeon, a big pigeon has 1 rough core and is severely excited, we can infer that the big pigeon has (1 + 2) = 3 rough cores.
[Step 3]
Rule: If A is a huge fairy, it has 10 luminous mouths. If A is completely buried, A will drop 3 luminous mouths.
Since A is a huge fairy, a huge fairy has 10 luminous mouths and is completely buried, we can infer that the huge fairy has (10 - 3) = 7 luminous mouths.
[Step 4]
Rule: If A is a medium griffin, it has 7 broken scales. If A is severely transformed, A will drop 2 broken scales.
Since A is a medium griffin, a medium griffin has 7 broken scales and is severely transformed, we can infer that the medium griffin has (7 - 2) = 5 broken scales.
Therefore, the final answer is \boxed{6, 3, 7, 5}.

Multi-Hop Rule:
Answer the following question based on the RULE LIBRARY:
[QUESTION]
How many muddy livers does the medium duck have after the medium duck is believed by a huge cat?
Let's think step by step and put your final answer within \boxed{}.

Based on the rules provided:
[Step 1]
Rule: If A believes B, B will enter royal court.
Since A is a huge cat, B is a medium duck, the huge cat believes the medium duck, we can infer that the medium duck will be in royal court.
[Step 2]
Rule: If A is in royal court, A will be severely depressed.
Since A is a medium duck and is in royal court, we can infer that it is severely depressed.
[Step 3]
Rule: If A is severely depressed, A will get 2 eternal souls.
Since A is a medium duck and is severely depressed, we can infer that the medium duck will get 2 eternal souls.
[Step 4]
Rule: If A is a medium duck, it has 10 muddy livers. If A gains 1 eternal soul, A will get 3 muddy livers.
Since A is a medium duck, a medium duck has 10 muddy livers, the medium duck gets 2 eternal souls, we can infer that the medium duck has (10 + 2 * 3) = 16 muddy livers.
Therefore, the final answer is \boxed{16}.

Figure 23: Examples of Single-Rule, Parallel Multi-Rule, and Multi-Hop Rule QA under NL representation.

Method	Single-Rule QA	Parallel Multi-Rule QA							Multi-Hop Rule QA			Avg.
	single-rule	multi-rule-2	multi-rule-3	multi-rule-4	multi-rule-5	multi-rule-6	multi-rule-7	multi-rule-8	multi-hop-2	multi-hop-3	multi-hop-4	
FOL Representation												
Rule Num = 100												
Qwen2.5-14B-Instruct	0.9200	0.8300	0.8400	0.7633	0.7483	0.6450	0.6950	0.6250	0.6400	0.1600	0.1000	0.6333
Qwen2.5-32B-Instruct	0.9600	0.9900	0.9000	0.8433	0.8267	0.7433	0.6950	0.6700	0.6000	0.2600	0.1400	0.6935
Qwen2.5-72B-Instruct	0.9200	0.9700	0.9433	0.9250	0.8783	0.8117	0.8950	0.8850	0.7400	0.6000	0.4800	0.8226
Llama-3.1-70B-Instruct	0.9800	0.9600	0.9567	0.8183	0.8683	0.8033	0.8250	0.8300	0.8000	0.4600	0.4400	0.7947
Llama-3.3-70B-Instruct	0.9600	0.9200	0.8467	0.7983	0.8333	0.7900	0.7900	0.8250	0.8600	0.5200	0.2400	0.7621
Qwen3-32B	0.8000	0.9500	0.9133	0.8950	0.9267	0.8917	0.8050	0.7850	0.8800	0.7800	0.4400	0.8242
deepseek-chat	0.8600	0.9400	0.9433	0.8733	0.9517	0.9017	0.9000	0.9300	0.9200	0.4400	0.2600	0.8109
gpt-5.1-2025-11-13	0.9600	0.9100	0.9233	0.8750	0.9017	0.8583	0.9000	0.9450	0.7400	0.3000	0.1800	0.7721
gpt-5.5-2026-04-23	0.9800	0.9200	0.9500	0.9750	0.8100	0.9700	**0.9900**	0.9500	0.7300	0.3300	0.3100	0.8105
claude-sonnet-4-6	0.9200	0.6200	0.9500	**0.9850**	0.8200	**0.9950**	0.9800	0.9500	0.5200	**0.9750**	0.6250	0.8491
DynaRule	**1.0000**	**1.0000**	**0.9950**	0.9800	**0.9950**	0.9850	0.9700	**0.9950**	**0.9600**	0.9300	**0.9500**	**0.9782**
Rule Num = 500												
Qwen2.5-14B-Instruct	0.8000	0.8500	0.8300	0.7617	0.5633	0.4283	0.3000	0.1450	0.4200	0.0400	0.0000	0.4671
Qwen2.5-32B-Instruct	0.8200	0.8800	0.8333	0.7600	0.6700	0.5050	0.3350	0.2000	0.5400	0.0800	0.0400	0.5148
Qwen2.5-72B-Instruct	0.7600	0.9500	0.9167	0.8566	0.7933	0.6350	0.5050	0.3250	0.6800	0.2600	0.1600	0.6220
Llama-3.1-70B-Instruct	0.9000	0.8500	0.8300	0.7283	0.7617	0.6017	0.5050	0.3050	0.5200	0.2400	0.1200	0.5783
Llama-3.3-70B-Instruct	0.9000	0.8600	0.8500	0.6833	0.6167	0.4000	0.3250	0.1800	0.7000	0.0200	0.0800	0.5105
Qwen3-32B	0.7000	0.8100	0.7900	0.6950	0.6150	0.4617	0.3450	0.3350	0.7800	0.3800	0.3000	0.5647
deepseek-chat	0.7800	0.9400	0.9500	0.8333	0.8617	0.8067	0.8300	0.7750	0.8200	0.2800	0.1600	0.7306
gpt-5.1-2025-11-13	0.9000	0.9300	0.8933	0.8600	0.8183	0.7800	0.7800	0.7450	0.6800	0.2200	0.1000	0.7006
gpt-5.5-2026-04-23	0.9800	0.9250	0.9550	0.9750	0.8100	0.8000	0.9000	0.7750	0.6200	0.3200	0.2100	0.7518
claude-sonnet-4-6	0.9100	0.7250	0.9500	0.9850	0.8250	0.9250	**0.9750**	**0.9850**	0.4300	**0.7400**	0.4400	0.8082
DynaRule	**1.0000**	**1.0000**	**0.9867**	**0.9950**	**0.9833**	**1.0000**	0.9650	**0.9850**	**0.9800**	0.7200	**0.8000**	**0.9468**
Rule Num = 1000												
Qwen2.5-14B-Instruct	0.7200	0.6900	0.6333	0.7150	0.5266	0.2950	0.1900	0.0900	0.5400	0.0200	0.0200	0.4036
Qwen2.5-32B-Instruct	0.8400	0.7900	0.7167	0.7100	0.6200	0.3600	0.2200	0.1100	0.5400	0.0800	0.0000	0.4533
Qwen2.5-72B-Instruct	0.7200	0.7700	0.8067	0.8483	0.7067	0.5000	0.3700	0.1600	0.6000	0.1000	0.0600	0.5129
Llama-3.1-70B-Instruct	0.8600	0.8000	0.8133	0.6983	0.6400	0.3983	0.3600	0.1900	0.4600	0.1000	0.0600	0.4891
Llama-3.3-70B-Instruct	0.8600	0.7000	0.6833	0.6517	0.6083	0.2750	0.2200	0.0500	0.4600	0.0400	0.0000	0.4135
Qwen3-32B	0.5800	0.7100	0.7834	0.7117	0.6483	0.4100	0.3300	0.1350	0.5600	0.1200	0.1000	0.4626
deepseek-chat	0.8800	0.9200	0.9033	0.7817	0.8250	0.7050	0.6950	0.6450	0.7800	0.1400	0.1200	0.6723
gpt-5.1-2025-11-13	0.9600	0.8800	0.8933	0.7867	0.7733	0.6950	0.6850	0.5900	0.7400	0.1400	0.0800	0.6567
gpt-5.5-2026-04-23	0.9800	0.9100	0.9550	0.9250	0.7000	0.8500	0.7750	0.9000	0.5300	0.3150	0.2150	0.7323
claude-sonnet-4-6	0.9000	0.8300	0.9500	0.9650	0.7750	0.9250	0.9750	0.9620	0.4100	**0.7200**	0.4100	0.8020
DynaRule	**1.0000**	**1.0000**	**0.9917**	**0.9875**	**0.9792**	**0.9625**	**0.9812**	**0.9688**	**0.8500**	0.6250	**0.6500**	**0.9087**
NL Representation												
Rule Num = 100												
Qwen2.5-14B-Instruct	0.9000	0.8600	0.9067	0.8217	0.7900	0.7333	0.6900	0.7350	0.7400	0.2200	0.3800	0.7070
Qwen2.5-32B-Instruct	0.9400	0.9300	0.9333	0.8633	0.9084	0.7767	0.8400	0.7600	0.7600	0.2400	0.2400	0.7447
Qwen2.5-72B-Instruct	0.8800	0.9100	0.9600	0.9133	0.9717	0.9133	0.9250	0.9200	0.7600	0.5400	0.5400	0.8394
Llama-3.1-70B-Instruct	0.9600	0.8700	0.8567	0.8500	0.8983	0.7317	0.8300	0.9050	0.5200	0.4200	0.5600	0.7638
Llama-3.3-70B-Instruct	0.9800	0.9200	0.9433	0.7800	0.8950	0.8150	0.7950	0.8100	0.7000	0.4400	0.4800	0.7780
Qwen3-32B	0.8400	0.9500	0.9500	0.9200	0.9250	0.8017	0.8250	0.7950	0.7800	0.7800	0.7600	0.8479
deepseek-chat	0.9000	0.9600	0.9333	0.8950	0.9500	0.8783	0.8700	0.9300	**0.9600**	0.7200	0.6600	0.8779
gpt-5.1-2025-11-13	0.9800	0.9300	0.9400	0.9000	0.9083	0.8517	0.9100	0.8900	0.8600	0.3600	0.3600	0.8082
gpt-5.5-2026-04-23	**1.0000**	0.9000	0.9500	**1.0000**	0.7500	0.9750	0.9500	0.8750	0.7000	0.9000	0.6000	0.8727
claude-sonnet-4-6	**1.0000**	0.8200	0.9500	0.9667	0.8000	0.9500	**0.9750**	**1.0000**	0.8000	0.9000	**0.9000**	0.9147
DynaRule	**1.0000**	**1.0000**	**0.9950**	0.9950	**0.9900**	**0.9800**	**0.9750**	0.9850	0.9200	**0.9600**	0.7000	**0.9545**
Rule Num = 500												
Qwen2.5-14B-Instruct	0.8000	0.6500	0.7000	0.6233	0.4850	0.3650	0.2100	0.1150	0.6600	0.0200	0.0200	0.4226
Qwen2.5-32B-Instruct	0.8800	0.8600	0.8133	0.7900	0.7067	0.5133	0.4100	0.3700	0.6000	0.0600	0.0800	0.5530
Qwen2.5-72B-Instruct	0.7200	0.9000	0.8700	0.8067	0.8350	0.7050	0.6850	0.6700	0.5000	0.2200	0.1400	0.6411
Llama-3.1-70B-Instruct	**1.0000**	0.9100	0.8733	0.7967	0.7267	0.5917	0.5000	0.5850	0.5400	0.1400	0.1200	0.6167
Llama-3.3-70B-Instruct	**1.0000**	0.8500	0.8433	0.7483	0.5667	0.4817	0.3950	0.3250	0.6000	0.0600	0.0600	0.5391
Qwen3-32B	0.8000	0.7400	0.7633	0.7117	0.5850	0.4550	0.3500	0.3250	0.7200	0.3400	0.5000	0.5718
deepseek-chat	0.8600	0.9200	0.8933	0.8450	0.8400	0.7617	0.7250	0.6900	0.8200	0.4200	0.3800	0.7414
gpt-5.1-2025-11-13	0.9600	0.9100	0.9200	0.8866	0.8467	0.7433	0.7400	0.7600	0.8200	0.1800	0.2000	0.7242
gpt-5.5-2026-04-23	**1.0000**	0.9000	0.9500	**1.0000**	0.8000	0.9000	0.8500	0.9750	0.5000	**0.9000**	0.5000	0.8432
claude-sonnet-4-6	0.8900	0.8000	0.9500	**1.0000**	0.7667	0.9500	**1.0000**	**1.0000**	0.6000	**0.9000**	0.8000	0.8779
DynaRule	0.9800	**0.9800**	**0.9950**	**1.0000**	**0.9783**	**0.9800**	0.9600	0.9750	**0.8800**	0.7000	**0.8200**	**0.9317**
Rule Num = 1000												
Qwen2.5-14B-Instruct	0.6200	0.6000	0.6067	0.6533	0.4600	0.3117	0.1450	0.0450	0.5600	0.0200	0.0000	0.3656
Qwen2.5-32B-Instruct	0.7400	0.8200	0.7700	0.7767	0.5683	0.4050	0.3050	0.1050	0.6000	0.0400	0.0200	0.4682
Qwen2.5-72B-Instruct	0.7600	0.8000	0.7300	0.8533	0.7100	0.6100	0.5400	0.4450	0.4400	0.0800	0.0400	0.5462
Llama-3.1-70B-Instruct	0.9600	0.7800	0.7800	0.7200	0.6600	0.4933	0.2950	0.2150	0.4000	0.0000	0.0400	0.4858
Llama-3.3-70B-Instruct	0.9600	0.6900	0.7467	0.7050	0.5083	0.3400	0.2250	0.0650	0.5600	0.0000	0.0000	0.4364
Qwen3-32B	0.8800	0.7000	0.7433	0.6433	0.5133	0.3733	0.3600	0.1200	0.6400	0.1800	0.1800	0.4848
deepseek-chat	0.9000	0.8700	0.8567	0.7883	0.7600	0.6450	0.6000	0.5450	0.8000	0.1800	0.1600	0.6459
gpt-5.1-2025-11-13	**1.0000**	0.8900	0.8833	0.8717	0.7583	0.6583	0.6250	0.6350	0.7000	0.1000	0.0400	0.6511
gpt-5.5-2026-04-23	0.8000	0.9000	0.9500	**1.0000**	0.7000	**1.0000**	0.8750	**0.9500**	0.8000	0.4000	0.2000	0.7795
claude-sonnet-4-6	0.8300	0.7000	0.9500	**1.0000**	0.7417	0.8250	0.8750	0.9250	0.3300	**0.8100**	**0.7900**	0.7979
DynaRule	0.9800	**0.9800**	**0.9817**	0.9883	**0.9783**	0.9783	**0.9550**	0.9450	**0.8400**	0.6400	0.6800	**0.9042**

Table 13: Prompting comparison results on the RuleWorld benchmark under the **FOL** and **NL** rule representations, compared with DynaRule.