FastLog: Efficient End-to-end Rule Learning Over Large-scale Knowledge Graphs by Reduction to Vector Operations

Anonymous ACL submission

Abstract

Logical rules play a crucial role in the evolution of knowledge graphs (KGs), as they can infer new facts from existing ones while providing explanations. In recent years, end-to-end rule learning has emerged as a promising paradigm to learn logical rules. The key insight of end-to-end rule learning is to transform the rule learning problem in a discrete space into the parameter learning problem in a continuous space, by employing TensorLog operators to simulate the inference of logical rules. However, these TensorLog-based methods struggle with limited scalability in learning rules from large-scale KGs. To improve the efficiency and scalability of end-to-end rule learning, we propose an efficient framework named FastLog for reducing vector-matrix multiplications to vector computations. FastLog is proven to have a lower time complexity than TensorLog. Extensive experimental results on a variety of benchmark KGs demonstrate that FastLog improves the efficiency of end-to-end methods by a significant margin without efficacy degradation in link prediction. Notably, by enhancing with FastLog, existing end-to-end methods are enabled to learn logical rules on two large-scale datasets with up to three hundred million triples, while achieving a high efficacy comparable with the most advanced rule learner within the same training time.

1 Introduction

Knowledge graph (KG) is a popular formalism to store real-world facts. Nowadays KGs have been widely employed in many real-world applications, including knowledge-based question answering (Mitra and Baral, 2016), recommendation (Lyu et al., 2020), information retrieval (Xiong et al., 2017) etc. A KG is usually represented as a directed graph where vertices are labeled by entities and edges by relations. A *fact* (also called a *triple*) in a KG is of the form (h, r, t), where h denotes the *head* entity, r the *relation* and t the *tail* entity. By now large-scale KGs such as YAGO (Suchanek et al., 2007), DBpedia (Auer et al., 2007) and Wikidata (Vrandecic and Krötzsch, 2014) consist of hundreds of millions of facts, underpinning various downstream applications.

Logical rules are pivotal in KG reasoning. They can infer new facts from existing ones and excel in explaining why the new facts are inferred. In recent years, end-to-end rule learning (Yang et al., 2017; Sadeghian et al., 2019; Wang et al., 2024b; Qi et al., 2023) becomes a popular paradigm for learning logical rules. The key insight of end-to-end methods is to convert the predicate selection problem in a discrete space into the parameter learning problem in a continuous space. This conversion enables end-to-end learning of logical rules from noisy data (Yang et al., 2017; Ye et al., 2023).

End-to-end methods such as NeuralLP (Yang et al., 2017) and DRUM (Sadeghian et al., 2019) usually exploit TensorLog (Cohen et al., 2020) operators to simulate the inference of logical rules. Specifically, TensorLog leverages a set of adjacency matrices to represent the background KG, where each adjacency matrix stores triples with the same relation. These matrices are then used to simulate the inference of logical rules. Figure 1 (a) illustrates an example of the calculation processes for TensorLog operators, where both M_{r_1}, $M_{r_1^-}$ and M_I are sparse matrices. We can observe that the number of floating-point multiplications and additions for TensorLog in this example is $3|\mathcal{K}| + (2|\mathcal{R}| + 1)|\mathcal{E}| = 30$, where $|\mathcal{K}|$ denotes the number of non-zero elements in all sparse matrices, $|\mathcal{E}|$ (resp. $|\mathcal{R}|$) denotes the total number of entities (resp. relations). In practice, $(2|\mathcal{R}| + 1)|\mathcal{E}|$ is particularly large in real-world KGs, e.g., $(2|\mathcal{R}| + 1)|\mathcal{E}| = 2.6e12$ for the Freebase (Kochsiek and Gemulla, 2021) dataset. Such a huge amount of computation may impair the

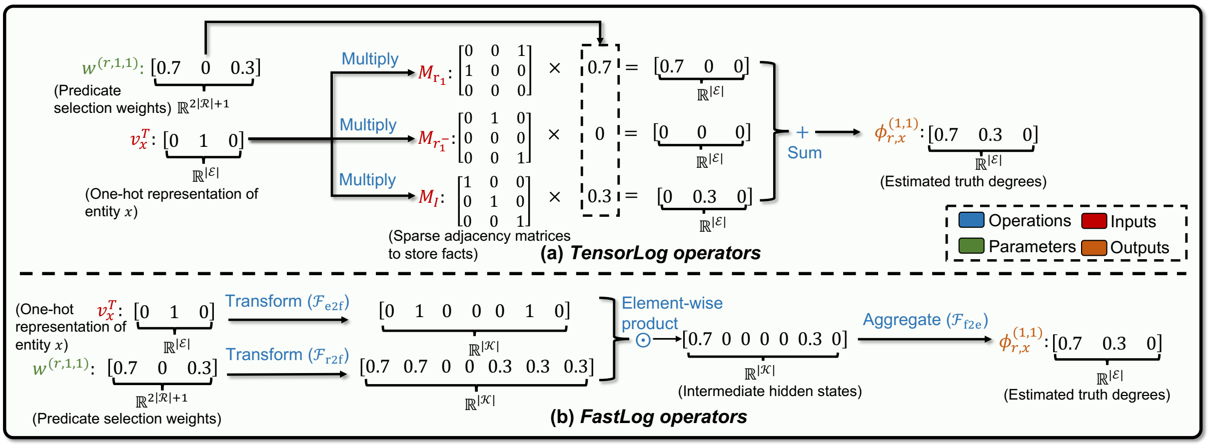


Figure 1: Examples of the calculation processes for TensorLog and FastLog.

scalability of end-to-end methods on large-scale KGs. As pointed out by Meilicke et al. (2024), by now there is no end-to-end approach that has evaluation reports on large-scale KGs such as Wikidata5M (Wang et al., 2021) and Freebase.

To enable practical end-to-end learning of logical rules from large-scale KGs, we propose a novel computational framework named FastLog. It introduces a sequence of vector operators to simulate the inference of logical rules. These vector operators reduce vector-matrix multiplications into vector computations, thereby considerably decreasing the time complexity. Figure 1 (b) illustrates an example of the calculation processes for FastLog operators, where both $\mathcal{F}_{\mathrm{e2f}}$, $\mathcal{F}_{\mathrm{r2f}}$ and $\mathcal{F}_{\mathrm{f2e}}$ are vector-based operators designed in FastLog. We can observe that the number of floating-point multiplications and additions for FastLog in this example is $2|\mathcal{K}| = 21 < 30$. This reveals that FastLog has a lower computation cost than TensorLog in real-world KGs. For example, we have $(2|\mathcal{R}|+1)|\mathcal{E}| = 2.6e12 \gg 2|\mathcal{K}| = 1.5e9$ on the Freebase dataset.

Furthermore, we introduce a dynamic pruning strategy to further reduce the time complexity of the backward propagation steps for FastLog. This strategy omits intermediate hidden states that have relatively low impacts on the reasoning process. By applying this dynamic pruning strategy, we show in Proposition 6 that the time complexity of a backward propagation step in FastLog can be further reduced to a constant. Thanks to the relatively low time complexity of FastLog, existing methods enhanced by FastLog become capable of learning rules from very large KGs (e.g., Freebase) with limited time cost (e.g., several hours).

We apply FastLog to enhancing four state-of-the-art (SOTA) end-to-end methods, including NeuralLP, DRUM, smDRUM (Wang et al., 2024b), and mmDRUM (Wang et al., 2024b). We empirically evaluate the original methods and the FastLog-enhanced ones for link prediction on totally ten benchmark KGs, among which six are relatively small, two are large-scale, and two are under the inductive setting. Experimental results on the six relatively small KGs demonstrate that the four FastLog-enhanced methods achieve 2.5x to 50x speedups compared to their original methods, while keeping almost the same efficacy in link prediction. For the two large-scale KGs, the FastLog-enhanced methods exhibit comparable efficacy in link prediction as the currently most advanced search-based method AnyBURL (Meilicke et al., 2024), by spending the same training time. For two datasets under the inductive setting, the FastLog-enhanced methods significantly outperform AnyBURL by a significant margin in the link prediction task.

2 Related Work

Learning logical rules from knowledge bases (KBs) has been widely studied in the field of Inductive Logic Programming (ILP) (Muggleton and Raedt, 1994; Zeng et al., 2014), where logical rules are learnt in a generate-and-test manner. This manner is a two-step pipeline that first generates logical rules from relational paths in a KB, and then filters rules with high confidence scores based on some tests. Modern ILP methods like AMIE+ (Galárraga et al., 2015) and AnyBURL (Meilicke et al., 2024) learn logical rules from KGs based on the Closed-World Assumption (CWA) (Galárraga et al., 2013). More specifically, they treat triples outside a KG as negative examples and exploit various search heuristics to efficiently learn rules. They are called *search-based* methods due to the use of heuristic search strategies. According to the empirical study conducted by Meilicke et al. (2024), only AnyBURL among the above search-based methods demonstrates the capability to learn logical rules from KGs containing more than 100 million facts. However, AnyBURL requires massive main memory (e.g., 900GB RAM for Freebase) to build up index structures for accelerating reasoning. In contrast, all FastLog-enhanced methods can run within 25GB RAM in conjunction with a single NVIDIA 4090 GPU with 24GB memory.

More recently, there has been an emerging interest in exploiting end-to-end methods (Yang et al., 2017; Sadeghian et al., 2019; Qi et al., 2023; Wang et al., 2024b) for rule learning. They usually parameterize a neural network to simulate the inference of logical rules by employing TensorLog (Cohen et al., 2020) operators and tune the parameters of the neural network by gradient descent. Thanks to the end-to-end learning manner, these methods work well with imperfect data (Yang et al., 2017) in learning logical rules. Despite their successes, the scalability of end-to-end methods is still limited. As far as we know, there is no end-to-end method that can perform rule learning on large-scale KGs. To facilitate existing end-to-end methods to learn logical rules from large-scale KGs, we propose an efficient framework named FastLog that has a lower time complexity than TensorLog.

3 Preliminaries

Knowledge graph. Let $\mathcal{E}$ be a set of entities and $\mathcal{R}$ a set of relations, a knowledge graph $\mathcal{G}$ is a subset of $\mathcal{E}\times\mathcal{R}\times\mathcal{E}$. Specifically, $\mathcal{G} = \{(h_i, r_i, t_i)\}_{1\leq i\leq N}$, where N denotes the number of triples, $h_i \in \mathcal{E}$ the *head* entity for the i^{th} triple, $r_i \in \mathcal{R}$ the relation for the i^{th} triple and $t_i \in \mathcal{E}$ the *tail* entity for the i^{th} triple. By r^- we denote the inverse relation of $r \in \mathcal{R}$. The set of inverse relations for $\mathcal{R}$, namely $\{r^- \mid r \in \mathcal{R}\}$, is denoted by $\mathcal{R}^-$. Accordingly, the equivalent knowledge graph for $\mathcal{G}$ composed by inverse relations, namely $\{(t, r^-, h) \mid (h, r, t) \in \mathcal{G}\}$, is denoted by $\mathcal{G}^-$.

Chain-like rule. An *atom* is a basic first-order logic formula of the form $p(u_1, \ldots, u_n)$, where p is a *predicate* and $u_1, \ldots, u_n$ are terms that denote either constants or variables. An r-specific *chain-like rule* (CR) R for is of the form:

$$r(x, y) \leftarrow p_1(x, z_1) \wedge p_2(z_1, z_2) \wedge \ldots \wedge p_L(z_{L-1}, y),$$

where x (resp. y) is the head (resp. tail) entity variable, $z_1, \ldots, z_{L-1}$ are other variables, $p_1, \ldots, p_L$ are predicates, and r denotes the predicate of a new fact that inferred by a r-specific CR. The part of R at the left (resp. right) of $\leftarrow$ is called the *head* (resp. *body*) of R. To uniformly represent r-specific CRs using fixed-length bodies, DRUM (Sadeghian et al., 2019) introduces the *identity relation* (denoted by I) to rule bodies. For example, $r(x, y) \leftarrow p(x, y)$ can be converted into a rule with two body atoms, namely $r(x, y) \leftarrow p(x, z) \wedge I(z, y)$.

TensorLog operators. End-to-end methods (Yang et al., 2017; Sadeghian et al., 2019; Wang et al., 2024b) aim to convert the discrete rule learning problem into a parameterized optimization problem in a continuous space, where the learnt rules are extracted from their parameter assignments. The TensorLog (Cohen et al., 2020) framework is the foundation for end-to-end rule learning to simulate the inference of CRs. We elaborate on the formalization of TensorLog as follows.

Suppose $\mathcal{R} = \{r_i\}_{1\leq i\leq n}$, its corresponding set of inverse relations $\mathcal{R}^- = \{r_i\}_{n+1\leq i\leq 2n}$ and $I = r_{2n+1}$, where n denotes the number of relations and $r_{i+n} = r_i^-$ for all $1 \leq i \leq n$. Let $\mathcal{G}$ be a knowledge graph. TensorLog first represents the background knowledge $\mathcal{K} = \mathcal{G} \cup \mathcal{G}^- \cup \{I(e, e) \mid e \in \mathcal{E}\}$ by a set of sparse adjacency matrices $\{M_{r_i}\}_{1\leq i\leq 2n+1}$, where $M_{r_i} \in \{0, 1\}^{|\mathcal{E}|\times|\mathcal{E}|}$ is a sparse adjacency matrix to store the set of triples $\{(h, r_i, t) \in \mathcal{K}\}$. For an arbitrary triple $(x, r, y) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, given the maximum number N of rules to be learnt, the maximum length L of each rule and a set of trainable parameters $\theta_r^{N,L} = \{w_i^{(r,k,l)}\}_{1\leq k\leq N, 1\leq l\leq L, 1\leq i\leq 2n+1}$ for r, for all $1 \leq k \leq N, 1 \leq l \leq L$, the intermediate truth degrees $\phi_{r,x}^{(k,l)} \in \mathbb{R}^{|\mathcal{E}|}$ are estimated by

$$\phi_{r,x}^{(k,l)} = \sum_{i=1}^{2n+1} \phi_{r,x}^{(k,l-1)}(w_i^{(r,k,l)} M_{r_i}), \quad (1)$$

where $\phi_{r,x}^{(k,0)} = v_x^\top$, $v_x \in \{0, 1\}^{|\mathcal{E}|}$ denotes the one-hot representation of entity x. $w^{(r,k,l)} \in [0, 1]^{2n+1}$ denotes the predicate selection weights, and it is confined to $[0, 1]^{2n+1}$ by a softmax layer. The truth degree of (x, r, y) is calculated by

$$\texttt{TensorLog}(\theta_r^L, x, y) = \sum_{k=1}^{N} \phi_{r,x}^{(k,L)} v_y, \quad (2)$$

where $v_y \in \{0, 1\}^{|\mathcal{E}|}$ denotes the one-hot representation of entity y. Intuitively, the estimated truth degree $\texttt{TensorLog}(\theta_r^{N,L}, x, y)$ reflects the degree of whether the triple (x, r, y) can be inferred by a certain rule among N-CRs.

Throughout the paper, we consider the worst-case time complexity for the training phases of end-to-end methods, where the time complexity is derived based on the number of floating-point multiplications and additions. The following Proposition 1[1] shows the time complexity of TensorLog.

Proposition 1. *Let $\mathcal{K} = \mathcal{G} \cup \mathcal{G}^- \cup \{I(e, e) \mid e \in \mathcal{E}\}$. The time complexity of a forward computation step for* TensorLog *is $\mathcal{O}(NL(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|))$. The time complexity of a backward propagation step for* TensorLog *is $\mathcal{O}(NL(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|))$.*

4 The FastLog Framework

To reduce the time complexity of TensorLog, we propose FastLog, an efficient framework to scale existing end-to-end approaches to learn rules on large-scale KGs. The intuition of FastLog is to convert the sparse vector-matrix multiplications used in TensorLog into vector operations, thereby improving efficiency. Figure 1 illustrates examples of the calculation processes for TensorLog and FastLog. It can be seen that a sequence of vector-matrix multiplications used in TensorLog is equivalent to several steps of vector operations in FastLog. Besides, we find that vector operations in FastLog have both lower time cost and lower space cost than vector-matrix multiplications in TensorLog when the proportion of non-zero elements in sparse matrices is below a certain value. We will elaborate on this value in the discussions of Proposition 2 and Proposition 4.

To implement FastLog operators, we first introduce three vector-based functions. The first function, denoted by $\mathcal{F}_{\text{e2f}}$, maps the intermediate estimated truth degrees, represented as a $|\mathcal{E}|$-dimensional vector, to a $|\mathcal{K}|$-dimensional vector. Similarly, the second function, denoted by $\mathcal{F}_{\text{r2f}}$, maps the predicate selection weights, represented as a $(2|\mathcal{R}| + 1)$-dimensional vector, to a $|\mathcal{K}|$-dimensional vector. The third function, denoted by $\mathcal{F}_{\text{f2e}}$, aggregates a $|\mathcal{K}|$-dimensional vector to a $|\mathcal{E}|$-dimensional vector.

Suppose $\mathcal{K} = \mathcal{G} \cup \mathcal{G}^- \cup \{I(e, e) \mid e \in \mathcal{E}\} = \{\tau_j\}_{1 \le j \le 2|\mathcal{G}|+|\mathcal{E}|}$. For all $1 \le i \le |\mathcal{K}|$, $\mathcal{F}_{\text{e2f}} : \mathbb{R}^{|\mathcal{E}|} \to \mathbb{R}^{|\mathcal{K}|}$ is a function such that the i-th element of $\mathcal{F}_{\text{e2f}}(v)$ is

$$[\mathcal{F}_{\text{e2f}}(v)]_i = [v]_{\text{head}(\tau_i)}, \tag{3}$$

where $v \in \mathbb{R}^{|\mathcal{E}|}$ denotes the input vector, and $\text{head}(\tau)$ is a function that returns the index (in $[1, |\mathcal{E}|]$) of the head entity of the triple τ. For all $1 \le i \le |\mathcal{K}|$, $\mathcal{F}_{\text{r2f}} : \mathbb{R}^{2|\mathcal{R}|+1} \to \mathbb{R}^{|\mathcal{K}|}$ is a function such that the i-th element of $\mathcal{F}_{\text{r2f}}(v)$ is

$$[\mathcal{F}_{\text{r2f}}(v)]_i = [v]_{\text{rel}(\tau_i)}, \tag{4}$$

where $v \in \mathbb{R}^{2|\mathcal{R}|+1}$ denotes the input vector, and $\text{rel}(\tau)$ is a function that returns the index (in $[1, 2|\mathcal{R}| + 1]$) of the relation of the triple τ. Let $\text{tail}(\tau)$ be a function that returns the index (in $[1, |\mathcal{E}|]$) of the tail entity of the triple τ. For all $1 \le i \le |\mathcal{E}|$, $\mathcal{F}_{\text{f2e}} : \mathbb{R}^{|\mathcal{K}|} \to \mathbb{R}^{|\mathcal{E}|}$ is a function such that the i-th element of $\mathcal{F}_{\text{f2e}}(v)$ is

$$[\mathcal{F}_{\text{f2e}}(v)]_i = \sum_{j:\text{tail}(\tau_j)=i} v_j, \tag{5}$$

where $v \in \mathbb{R}^{|\mathcal{K}|}$ denotes the input vector. For an arbitrary triple $(x, r, y) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, for all $1 \le k \le N, 1 \le l \le L$, the intermediate truth degrees $\phi_{r,x}^{(k,l)} \in \mathbb{R}^{|\mathcal{E}|}$ are estimated by

$$\phi_{r,x}^{(k,l)} = \mathcal{F}_{\text{f2e}}(\mathcal{F}_{\text{e2f}}(\phi_{r,x}^{(k,l-1)}) \odot \mathcal{F}_{\text{r2f}}(w^{(r,k,l)})), \tag{6}$$

where $\phi_{r,x}^{(k,0)} = v_x^\top$, and $\odot$ denotes the element-wise product. The truth degree of (x, r, y) is estimated by

$$\texttt{FastLog}(\theta_r^{N,L}, x, y) = \sum_{k=1}^{N} \phi_{r,x}^{(k,L)} v_y, \tag{7}$$

where $\theta_r^{N,L} = \{w_i^{(r,k,l)}\}_{1 \le k \le N, 1 \le l \le L, 1 \le i \le 2n+1}$ is a set of trainable parameters for the head relation r. The following Proposition 2 illustrates the time complexity of FastLog.

Proposition 2. *The time complexity of a forward computation step for* FastLog *is $\mathcal{O}(NL|\mathcal{K}|)$. The time complexity of a backward propagation step for* FastLog *is $\mathcal{O}(NL|\mathcal{K}|)$.*

The time complexity of FastLog is generally lower than TensorLog, as it holds that $|\mathcal{K}| \ll |\mathcal{R}||\mathcal{E}|$ in many real-world scenarios. For example, the Freebase (Kochsiek and Gemulla, 2021) dataset has $|\mathcal{K}|$=338M, $|\mathcal{E}|$=86M and $|\mathcal{R}|$=15k. It holds that $|\mathcal{R}||\mathcal{E}| \gg |\mathcal{K}|$. The following Proposition 3 demonstrates the correctness of FastLog.

[1] All proofs of this work are moved to Appendix B.

Proposition 3. *For an arbitrary triple* $(a, r, b) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, $\forall N \geq 1, L \geq 1$: $\texttt{TensorLog}(\theta_r^{N,L}, a, b) = \texttt{FastLog}(\theta_r^{N,L}, a, b)$.

Due to the space limitation, detailed formalizations of all `FastLog`-enhanced methods are moved to Appendix D.

4.1 Dynamic Pruning Strategy

Let m be the mini-batch size. The following Propositions 4-5 show the space complexity of `TensorLog` and that of `FastLog`, respectively.

Proposition 4. *The space complexity of a forward computation step for* `TensorLog` *is* $\mathcal{O}(m|\mathcal{E}|)$. *The space complexity of a backward propagation step for* `TensorLog` *is* $\mathcal{O}(mNL|\mathcal{R}||\mathcal{E}|)$.

Proposition 5. *The space complexity of a forward computation step for* `FastLog` *is* $\mathcal{O}(m|\mathcal{K}|)$. *The space complexity of a backward propagation step for* `FastLog` *is* $\mathcal{O}(mNL(|\mathcal{E}| + |\mathcal{K}|))$.

Note that the space complexity is derived from the number of floating-point numbers that must be stored during the reasoning process. From Propositions 3-4, we can infer that `FastLog` consumes less memory than `TensorLog` when $|\mathcal{K}| < \frac{(L|\mathcal{R}|+1)|\mathcal{E}|}{L+1}$. In general, we have $\mathcal{K} \ll \frac{(L|\mathcal{R}|+1)|\mathcal{E}|}{L+1}$ in most practical scenarios due to the sparsity of real-life KGs.

To further improve the efficiency, we propose a dynamic pruning strategy to further control the space complexity of a backward propagation step for `FastLog`. The intuition of this strategy is to filter out the reasoning paths that have relatively low impacts on reasoning. Given two hyper-parameters c_1 and c_2, we refine the functions $\mathcal{F}_{\text{e2f}}$, $\mathcal{F}_{\text{r2f}}$ and $\mathcal{F}_{\text{f2e}}$ to achieves this strategy. The refined version of $\mathcal{F}_{\text{e2f}}$, denoted by $\hat{\mathcal{F}}_{\text{e2f}}^{c_1}$, is defined as

$$\hat{\mathcal{F}}_{\text{e2f}}^{c_1}(\mathbb{T}) = \{(j, s) \mid (i, s) \in \mathcal{T}^{c_1}(\mathbb{T}), \text{head}(\tau_j) = i\}, \quad (8)$$

where $\mathbb{T}$ is a set of tuples. $\mathcal{T}^k(\mathbb{T})$ is a function that returns the top-k tuples from $\mathbb{T}$, where the tuples are ordered by the maximum value of their second elements. The refined version of $\mathcal{F}_{\text{r2f}}$, denoted by $\hat{\mathcal{F}}_{\text{r2f}}^{c_2}$, is defined as

$$\hat{\mathcal{F}}_{\text{r2f}}^{c_2}(\mathbb{T}, \omega) = \{(i, \omega_{\text{rel}(\tau_i)} s) \mid (i, s) \in \mathcal{T}^{c_2}(\mathbb{T})\}, \quad (9)$$

The refined version of $\mathcal{F}_{\text{f2e}}$, denoted by $\hat{\mathcal{F}}_{\text{f2e}}^{c_2}$, is defined as

$$\hat{\mathcal{F}}_{\text{f2e}}(\mathbb{T}) = \{(\text{tail}(\tau_i), \sum_{(i', s') \in \mathbb{T}, \text{tail}(\tau_{i'}) = \text{tail}(\tau_i)} s') \mid (i, s) \in \mathbb{T}\}, \quad (10)$$

For all $1 \leq k \leq N, 1 \leq l \leq L$, the set $\phi_{r,x}^{(k,l)}$ of intermediate truth degrees are estimated by

$$\hat{\phi}_{r,x}^{(k,l)} = \hat{\mathcal{F}}_{\text{f2e}}(\hat{\mathcal{F}}_{\text{r2f}}^{c_2}(\hat{\mathcal{F}}_{\text{e2f}}^{c_1}(\hat{\phi}_{r,x}^{(k,l-1)}), w^{(r,k,l)})), \quad (11)$$

where $\phi_{r,x}^{k,0} = \{(\mathcal{I}(x), 1)\}$ and $\mathcal{I}(e)$ is a function that returns the index of e (in $[1, |\mathcal{E}|]$). The truth degree of (x, r, y) is estimated by

$$\texttt{FastLog}^{c_1,c_2}(\theta_r^{N,L}, x, y) = \sum_{k=1}^{N} \sum_{(\mathcal{I}(y), s) \in \hat{\phi}_{r,x}^{(k,L)}} s. \quad (12)$$

The following Proposition 6 shows the time complexity of $\texttt{FastLog}^{c_1,c_2}$.

Proposition 6. *The time complexity of a forward computation step for* $\texttt{FastLog}^{c_1,c_2}$ *is* $\mathcal{O}(NL(|\mathcal{E}| + |\mathcal{K}|))$. *The time complexity of a backward propagation step for* $\texttt{FastLog}^{c_1,c_2}$ *is* $\mathcal{O}(NLc_2)$.

Note that `FastLog` uses the RadixSelect (Alabi et al., 2012) algorithm to implement the top-k function $\mathcal{T}^k$, which has a worst-case complexity of $\mathcal{O}(N)$ (Zhang et al., 2023), where N denotes the total number of elements. Proposition 6 reveals that we can control the time complexity of a backward propagation step for `FastLog` by setting c_2, where it can be that $c_2 \ll |\mathcal{K}|$ in practice. Propositions 7 shows the space complexity of $\texttt{FastLog}^{c_1,c_2}$.

Proposition 7. *The space complexity of a forward computation step for* $\texttt{FastLog}^{c_1,c_2}$ *is* $\mathcal{O}(m|\mathcal{K}|)$. *The space complexity of a backward propagation step for* $\texttt{FastLog}^{c_1,c_2}$ *is* $\mathcal{O}(mNL(c_1 + c_2))$.

Proposition 8 shows that `FastLog` amounts to a special case of $\texttt{FastLog}^{c_1,c_2}$.

Proposition 8. *Given a knowledge graph* $\mathcal{G}$, *for an arbitrary triple* $(a, r, b) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, $\forall N \geq 1, L \geq 1$: $\texttt{FastLog}(\theta_r^{N,L}, a, b) = \texttt{FastLog}^{|\mathcal{E}|,|\mathcal{K}|}(\theta_r^{N,L}, a, b)$.

5 Evaluation

5.1 Experimental Settings

Datasets. We conducted experiments in link prediction on six benchmark datasets, including Family (Yang et al., 2017), Kinship (Kok and Domingos, 2007), UMLS (Kok and Domingos, 2007), WN18RR (Dettmers et al., 2018), FB15k-237 (Toutanova and Chen, 2015) and YAGO3-10 (Suchanek et al., 2007). We also conducted experiments on two large KGs Wikidata5m (Wang et al., 2021) and Freebase (Kochsiek and Gemulla, 2021). Statistical details are reported in Table 1.

Dataset	$\|\mathcal{E}\|$	$\|\mathcal{R}\|$	$\|\mathcal{G}_{\text{train}}\|$	$\|\mathcal{G}_{\text{valid}}\|$	$\|\mathcal{G}_{\text{test}}\|$	$\|\mathcal{K}\|$
Family	3K	12	23.5K	2K	2.8K	50K
Kinship	104	25	3.2K	2.1K	5.3K	6.5K
UMLS	135	46	2K	1.3K	3.3K	4.1K
WN18RR	41K	11	87K	3K	3.1K	215K
FB15k-237	15K	237	272K	17K	20K	559K
YAGO3-10	123K	37	1,079K	5K	5K	2,281K
Wikidata5M	4,594K	822	20,625K	5.2K	5.3K	45,844K
Freebase	86,054K	15K	338,586K	10K	10K	763,226K

Table 1: Statistics of experimental datasets.

Type (Version)	FB15k-237					NELL-995				
	$\|\mathcal{E}\|$	$\|\mathcal{R}\|$	$\|\mathcal{G}_{\text{tra.}}\|$	$\|\mathcal{G}_{\text{val.}}\|$	$\|\mathcal{G}_{\text{test}}\|$	$\|\mathcal{E}\|$	$\|\mathcal{R}\|$	$\|\mathcal{G}_{\text{tra.}}\|$	$\|\mathcal{G}_{\text{val.}}\|$	$\|\mathcal{G}_{\text{test}}\|$
Train (V1)	1.6K	179	4.2K	489	492	3.1K	14	4.7K	414	439
Test (V1)	1.1K	179	2.0K	206	205	225	14	833	101	100
Train (V2)	2.6K	200	9.7K	1.2K	1.2K	2.6K	88	8.2K	922	968
Test (V2)	1.7K	200	4.1K	469	478	21K	88	4.6K	459	476
Train (V3)	3.7K	215	18K	22K	22K	4.6K	142	16K	1.9K	1.9K
Test (V3)	2.5K	215	7.4K	866	865	3.6K	142	8.0K	811	809
Train (V4)	4.7K	219	27K	3.4K	3.4K	2.1K	76	7.5K	876	867
Test (V4)	3.1K	219	12K	1.4K	1.4K	2.8K	76	7.1K	716	731

Table 2: Statistics of datasets for the inductive setting.

For a more comprehensive evaluation, we also conducted experiments on two datasets FB15k-237 (Teru et al., 2020) and NELL-995 (Teru et al., 2020) under the inductive setting. Note that these two datasets have four different versions corresponding to four different dataset splitting. Statistical details for all versions of the two datasets under the inductive setting are reported in Table 2.

Evaluation Metrics. For each test triple (h, r, t) in evaluation, we built two queries $(h, r, ?)$ and $(t, r^{-}, ?)$. We computed the truth degrees for corrupted tail triples and then computed the rank of the correct answer. Based on the rank, we reported the Mean Reciprocal Rank (MRR for short) and Hit@k (H@k for short) metrics under the filtered setting introduced by (Bordes et al., 2013). Following the work (Qu et al., 2021), the rank of the correct answer is defined by $j + (k + 1)/2$ in our evaluation setting, where j is the number of corrupted triples with higher truth degrees than the correct answer and k the number of corrupted triples with the same truth degree as the correct answer.

Implementation Details. We implemented FastLog [2] by Pytorch 2.4.0. All experiments were conducted on a Linux machine equipped with an Intel Xeon Gold 6338N CPU processor with 1TB RAM and an NVIDIA 4090 GPU with 24GB memory. Note that we require 1TB RAM to reproduce the results of AnyBURL, as AnyBURL requires 900GB RAM to learn rules from Freebase (Meilicke et al., 2024). FastLog only requires a maximum 25GB RAM for training and evaluation.

[2]Code and data are available at: link removed during double-blind reviewing.

5.2 Main Results

To reduce bias, we evaluated each method using five distinct random seeds {1, 12, 123, 1234, 12345}. For each metric, we report the mean scores based on five runs. Table 3 (resp. Table 4) reports the comparison results on Family, Kinship and UMLS (resp. WN18RR, FB15k-237 and YAGO3-10). Note that we did not apply the dynamic pruning strategy for Family, Kinship and UMLS due to their small size. Results show that the FastLog-enhanced methods achieve 2.5x to 50x speedups over their original methods. In particular, the original DRUM, smDRUM and mmDRUM methods cannot complete training on FB15k-237 within a limited time (1 day), while all original methods cannot complete training on YAGO3-10. In contrast, all FastLog-enhanced methods can complete training on FB15k-237 and YAGO3-10. These results confirm the high efficiency of FastLog. Furthermore, we can observe that only a few efficacy differences between the FastLog-enhanced methods and the original methods are statistically significant by two-tailed t-tests. These results demonstrate that FastLog keeps comparable efficacy of existing end-to-end methods. Besides, it can be seen that FastLog may spend slightly more GPU memory on Family, Kinship and UMLS, which is consistent with the space complexity results. Given the high efficiency achieved by FastLog, these slight increases in memory usage are acceptable.

Table 5 reports the comparison results on two large-scale datasets. Results show that all the original end-to-end methods cannot work on Wikidata5M and Freebase due to running out of memory (OOM). In contrast, the FastLog-enhanced methods can achieve comparable MRR scores on these datasets with the SOTA search-based method AnyBURL. This implies that FastLog is able to upgrade existing end-to-end methods to learn rules from large-scale KGs. Note that all FastLog-enhanced methods cannot outperform AnyBURL within the same training time (i.e., 20,000s). This may be because all FastLog-enhanced methods only learn chain-like rules for reasoning, whereas AnyBURL learns both chain-like rules and logical rules with entity constants. In general, learning more complex forms of rules may result in better efficacy in link prediction. To verify this, we created a variant of AnyBURL, denoted

Method	Family MRR	H@1	H@3	H@10	TT	MC
NeuralLP	0.923	87.1	97.2	98.7	448s	0.5GB
NeuralLP-FL	0.926	87.5	97.4	98.8	147s	0.7GB
Δ	(↑0.003)	(↑0.4)	(↑0.2)	(↑0.1)	(↑3.0x)	(↓0.2)
DRUM	0.941	89.8	98.2	99.0	565s	0.7GB
DRUM-FL	0.951	92.0	98.0	99.0	158s	1.1GB
Δ	(↑0.010)	(↑2.2)	(↓0.2)	(-)	(↑3.6x)	(↓0.4)
smDRUM	0.957	92.6	98.4	99.0	1119s	1.0GB
smDRUM-FL	0.959	93.0	98.4	99.0	190s	1.2GB
Δ	(↑0.002)	(↑0.4)	(-)	(-)	(↑5.9x)	(↓0.2)
mmDRUM	0.904	83.0	96.9	98.9	1072s	1.0GB
mmDRUM-FL	0.926	86.0	97.8	99.0	166s	1.2GB
Δ	(↑0.022*)	(↑3.0*)	(↑0.9*)	(↑0.1)	(↑6.5x)	(↓0.2)

Method	Kinship MRR	H@1	H@3	H@10	TT	MC
NeuralLP	0.468	30.4	54.7	82.6	73s	0.4GB
NeuralLP-FL	0.472	30.5	55.3	84.3	29s	0.6GB
Δ	(↑0.004)	(↑0.1)	(↑0.6)	(↑1.7*)	(↑2.5x)	(↓0.2)
DRUM	0.471	30.0	55.0	84.5	132s	0.5GB
DRUM-FL	0.475	30.4	55.5	85.5	37s	0.7GB
Δ	(↑0.004)	(↑0.4)	(↑0.5)	(↑1.0)	(↑3.6x)	(↓0.2)
smDRUM	0.425	25.1	49.8	82.1	303s	0.6GB
smDRUM-FL	0.439	26.3	51.5	84.2	40s	0.7GB
Δ	(↑0.014)	(↑1.2)	(↑1.7)	(↑2.1*)	(↑7.6x)	(↓0.1)
mmDRUM	0.286	13.0	30.7	66.8	214s	0.6GB
mmDRUM-FL	0.304	13.3	31.0	68.4	38s	0.7GB
Δ	(↑0.018)	(↑0.3)	(↑0.3)	(↑1.6)	(↑5.6x)	(↓0.1)

Method	UMLS MRR	H@1	H@3	H@10	TT	MC
NeuralLP	0.686	53.3	81.0	93.0	73s	0.4GB
NeuralLP-FL	0.707	55.1	84.1	93.6	19s	0.5GB
Δ	(↑0.021*)	(↑1.8)	(↑3.1*)	(↑0.6)	(↑3.8x)	(↓0.1)
DRUM	0.706	56.1	82.1	93.9	111s	0.5GB
DRUM-FL	0.742	60.3	86.3	94.7	22s	0.6GB
Δ	(↑0.036*)	(↑4.2*)	(↑4.2*)	(↑0.8*)	(↑5.0x)	(↓0.1)
smDRUM	0.738	60.1	84.8	94.3	179s	0.6GB
smDRUM-FL	0.744	61.4	85.0	94.4	30s	0.6GB
Δ	(↑0.006)	(↑1.3*)	(↑0.2)	(↑0.1)	(↑6.0x)	(-)
mmDRUM	0.465	31.8	52.0	79.3	221s	0.6GB
mmDRUM-FL	0.478	32.9	54.1	78.3	23s	0.6GB
Δ	(↑0.013)	(↑1.1)	(↑2.1)	(↓1.0)	(↑9.6x)	(-)

Table 3: Comparison results on Family, Kinship and UMLS, where TT abbreviates the training time, MC the memory cost on GPU and GB the Gigabytes. The differences marked by $*$ are statistically significant with p-value$<$0.05 by a two-tailed t-test. Δ denotes the performance difference.

Method	WN18RR MRR	H@1	H@3	H@10	TT	MC
NeuralLP	0.450	41.7	45.7	51.6	2.2h	2.5GB
NeuralLP-FL	0.450	41.7	45.7	51.9	551s	1.0GB
Δ	(-)	(-)	(-)	(↑0.3*)	(↑14.2x)	(↓1.5)
DRUM	0.459	42.2	47.1	53.3	2.2h	4.5GB
DRUM-FL	0.459	42.2	47.1	53.6	610s	1.5GB
Δ	(-)	(-)	(-)	(↑0.3*)	(↑13.2x)	(↓3.0)
smDRUM	0.410	35.6	43.2	51.7	4h	5.9GB
smDRUM-FL	0.421	37.4	43.7	51.4	546s	1.8GB
Δ	(↑0.011)	(↑1.8)	(↑0.5)	(↓0.3)	(↑26.5x)	(↓4.1)
mmDRUM	0.416	36.1	44.0	51.1	2.5h	5.9GB
mmDRUM-FL	0.420	37.0	44.2	51.2	556s	1.8GB
Δ	(↑0.004)	(↑0.9)	(↑0.2)	(↑0.1)	(↑16.5x)	(↓4.1)

Method	FB15k-237 MRR	H@1	H@3	H@10	TT	MC
NeuralLP	0.335	24.9	36.4	50.6	23h	8.6GB
NeuralLP-FL	0.334	24.9	36.4	50.8	0.7h	2.0GB
Δ	(↓0.1)	(-)	(↑0.1)	(↑0.2)	(↑32.6x)	(↓6.6)
DRUM	-	-	-	-	>1day	22.4GB
DRUM-FL	0.339	25.2	37.0	51.7	1.6h	4.4GB
Δ	(-)	(-)	(-)	(-)	(↑≈16x)	(↓18)
smDRUM	-	-	-	-	>1day	23.9GB
smDRUM-FL	0.280	18.7	30.5	46.4	1.7h	5.2GB
Δ	(-)	(-)	(-)	(-)	(↑≈16x)	(↓18.7)
mmDRUM	-	-	-	-	>1day	23.9GB
mmDRUM-FL	0.219	13.7	24.1	39.8	1.7h	4.2GB
Δ	(-)	(-)	(-)	(-)	(↑≈16x)	(↓19.7)

Method	YAGO3-10 MRR	H@1	H@3	H@10	TT	MC
NeuralLP	-	-	-	-	>1day	16.8GB
NeuralLP-FL	0.513	43.2	55.6	66.0	5.5h	5.9GB
Δ	(-)	(-)	(-)	(-)	(↑≈44x)	(↓10.9)
DRUM	-	-	-	-	>1day	22.4GB
DRUM-FL	0.431	35.4	47.5	58.1	9.6h	15.5GB
Δ	(-)	(-)	(-)	(-)	(↑≈50x)	(↓6.9)
smDRUM	-	-	-	-	>1day	23.0GB
smDRUM-FL	0.446	31.9	51.1	63.2	13.5h	13.7GB
Δ	(-)	(-)	(-)	(-)	(↑≈36x)	(↓9.3)
mmDRUM	-	-	-	-	>1day	23.3GB
mmDRUM-FL	0.365	24.6	41.4	55.7	11.6h	14.8GB
Δ	(-)	(-)	(-)	(-)	(↑≈42x)	(↓8.5)

Table 4: Comparison results on WN18RR, FB15k-237 and YAGO3-10, where TT abbreviates the training time, MC the memory cost on GPU and GB the Gigabytes. The differences marked by $*$ are statistically significant with p-value$<$0.05. Δ denotes the performance difference.

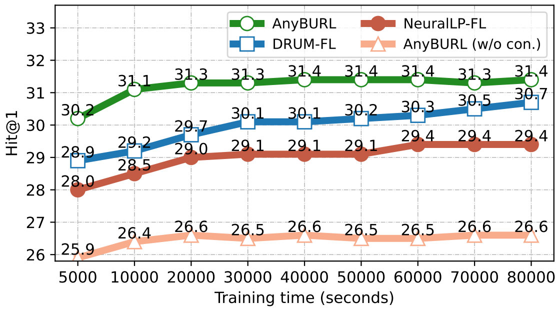


Figure 2: Comparison results for longer training time.

by AnyBURL (w/o constants), which only learns chain-like rules for reasoning. We can observe that the efficacy of AnyBURL significantly drops when only chain-like rules are learnt, and that DRUM-FL can outperform this variant in Hit@1 on both datasets. These results further affirm the effectiveness of the FastLog-enhanced methods. Besides, we show in Figure 2 that both NeuralLP-FL and DRUM-FL can benefit from more training time, whereas AnyBURL and its variant cannot achieve better efficacy by increasing the training time.

To verify the effectiveness of the proposed dynamic pruning strategy, we create a variant denoted by X-FL (w/o PS) for each FastLog-enhanced method by omitting the dynamic pruning strategy. We can observe that all variants cannot work on Freebase due to OOM. This indicates that the proposed dynamic pruning strategy is crucial for reducing the memory consumption of FastLog.

5.3 Discussions on Complexities and Results

From Proposition 1-2 we know that FastLog have a lower time cost than that of TensorLog, especially for those KGs that are relatively sparse (i.e., $|\mathcal{R}||\mathcal{E}| \gg |\mathcal{K}|$). Empirical results on Table 3-4 show that the FastLog-enhanced methods always have lower training time costs than that of their original methods, especially for sparse KGs like WN18RR, FB15k-237 and YAGO3-10. These findings align with the theoretical time complexity results we derived. Similarly, from Proposition 4-5 we know that FastLog demonstrates a lower memory cost than TensorLog when $|\mathcal{K}| < \frac{(L|\mathcal{R}|+1)|\mathcal{E}|}{L+1} \approx |\mathcal{R}||\mathcal{E}|$. This is consistent with the results presented in Tables 3-5, where FastLog-enhanced methods show higher memory costs on

Method	Wikidata5M							Freebase						
	MRR	H@1	H@3	H@10	TT	ET	MC	MRR	H@1	H@3	H@10	TT	ET	MC
AnyBURL (Meilicke et al., 2024)	**0.355**	**31.3**	**37.2**	**43.2**	20000s	18469s	-	**0.573**	**50.6**	**60.5**	**67.6**	20000s	9672s	-
AnyBURL (w/o constants)	0.304	26.6	31.8	36.4	20000s	20919s	-	0.544	47.7	57.6	64.4	20000s	10305s	-
NeuralLP	-	-	-	-	-	-	OOM	-	-	-	-	-	-	OOM
NeuralLP-FL	0.329	28.9	34.7	40.2	20000s	**337s**	**11.0GB**	0.537	47.5	56.8	63.7	20000s	8037s	**9.7GB**
NeuralLP-FL (w/o PS)	0.328	28.7	34.6	40.3	20000s	506s	19.2GB	-	-	-	-	-	-	OOM
DRUM	-	-	-	-	-	-	OOM	-	-	-	-	-	-	OOM
DRUM-FL	0.338	29.7	35.7	41.1	20000s	342s	13.0GB	0.544	48.0	57.5	64.6	20000s	**8029s**	11.5GB
DRUM-FL (w/o PS)	0.334	29.3	35.3	40.9	20000s	473s	20.4GB	-	-	-	-	-	-	OOM
smDRUM	-	-	-	-	-	-	OOM	-	-	-	-	-	-	OOM
smDRUM-FL	0.301	25.6	31.9	37.6	20000s	342s	13.4GB	0.530	45.7	56.3	64.1	20000s	8179s	11.4GB
smDRUM-FL (w/o PS)	0.297	25.3	31.4	37.1	20000s	499s	21.2GB	-	-	-	-	-	-	OOM
mmDRUM	-	-	-	-	-	-	OOM	-	-	-	-	-	-	OOM
mmDRUM-FL	0.278	23.2	29.6	35.5	20000s	347s	14.0GB	0.510	43.8	54.2	62.6	20000s	8185	11.4GB
mmDRUM-FL (w/o PS)	0.276	23.0	29.2	35.0	20000s	493s	21.2GB	-	-	-	-	-	-	OOM

Table 5: Comparison results on Wikidata5M and Freebase, where TT abbreviates the training time, MC the memory cost on GPU and GB the Gigabytes. The best value of each column has been highlighted.

Method	FB15k-237 (Inductive setting)								NELL-995 (Inductive setting)							
	V1		V2		V3		V4		V1		V2		V3		V4	
	MRR	H@1	MRR	H@1	MRR	H@1	MRR	H@1	MRR	H@1	MRR	H@1	MRR	H@1	MRR	H@1
AnyBURL($L = 3$)	0.366	30.5	0.477	36.8	0.447	33.4	0.424	31.6	0.734	67.5	0.438	32.9	0.373	28.9	0.362	20.7
AnyBURL($L = 6$)	0.369	30.2	0.458	34.8	0.449	34.1	0.430	31.9	0.633	47.5	0.435	31.4	0.371	28.9	0.364	21.1
AnyBURL† ($L = 3$)	0.362	30.0	0.476	37.2	0.447	33.4	0.429	31.7	0.723	65.5	0.446	33.1	0.359	27.2	0.369	21.8
AnyBURL† ($L = 6$)	0.364	29.5	0.373	36.3	0.408	30.3	0.427	31.7	0.611	44.5	0.431	32.9	0.362	28.1	0.363	21.3
DRUM-FL ($L = 3$)	0.416	34.4	0.514	41.7	0.489	39.4	**0.471**	**37.0**	**0.748**	**68.0**	**0.526**	**40.8**	0.485	**38.9**	0.384	25.7
DRUM-FL ($L = 6$)	**0.468**	**38.0**	**0.521**	**42.2**	**0.493**	**39.7**	0.469	36.9	0.671	57.0	0.501	38.1	**0.487**	**38.9**	**0.439**	**31.1**

Table 6: Comparison results on four versions of FB15k-237 and NELL-995 under the inductive setting.

datasets such as Family, Kinship, and UMLS, but significantly lower memory costs on larger datasets. Finally, from Proposition 6-7, we establish that FastLog can achieve a lower time and memory cost when the dynamic pruning strategy is applied and $c_2 < |\mathcal{K}|$. This is corroborated by the results in Table 5, where all FastLog-enhanced methods exhibit superior efficiency compared to their variants without the dynamic pruning strategy.

5.4 Inductive Setting

By comparing AnyBURL† (i.e., AnyBURL (w/o constants)) and AnyBURL, we know that the logical rules with entity constants contribute to the high efficacy of AnyBURL. Note that the logical rules with entity constants cannot generalize to the inductive setting where missing facts involve unseen entities. To verify this, we conducted experiments on four versions of FB15k-237 and NELL-995 under the inductive setting, as reported in Table 6. We followed the same inductive setting as the work (Teru et al., 2020), by using $\mathcal{G}_{\text{train}}$ in the training data for training and using $\mathcal{G}_{\text{test}}$ in the test data for evaluation, where the background knowledge for test is $\mathcal{G}_{\text{train}}$ in the test data. Results show that AnyBURL cannot benefit from the rules with entity constants on all datasets. Besides, we found that both AnyBURL and AnyBURL† cannot benefit from learning longer rules. In contrast, DRUM-FL benefits from learning longer rules on most datasets, significantly outperforming both AnyBURL and AnyBURL† on all datasets. These results reveal that learning logical rules with entity constants makes AnyBURL overfit the training data, resulting in limited efficacy under the inductive setting. In contrast, the FastLog-enhanced methods demonstrate better efficacy under the inductive setting thanks to their end-to-end learning manner. More analysis can refer to Appendix A.

6 Conclusion and Future Work

In this paper we have proposed an efficient framework named FastLog for end-to-end rule learning. We have proposed a novel vectorization optimization and a dynamic pruning strategy in FastLog to significantly reduce the time cost. Experimental results on six benchmark datasets demonstrate that the four FastLog-enhanced methods achieve 2.5x to 50x speedups compared to their original methods, while keeping comparable efficacy in link prediction. Furthermore, FastLog can upgrade four end-to-end methods to learn rules from two large-scale KGs that contain up to three hundred million triples. Future work will exploit FastLog to learn more complex logical rules for better efficacy.

7 Limitations

The major limitation of FastLog may be that all the FastLog-enhanced methods in this work learn chain-like rules only. In general, learning logical rules in a more complex form can help improve the efficacy for link prediction. For example, Table 5 shows that AnyBURL can benefit a lot from learning logical rules with entity constants. In practice, upgrading existing end-to-end methods to learn more complex rules is non-trivial. It requires well-designed neural modules to capture constraints on entity variables or on atoms, which is beyond the scope of this work. Therefore, we leave this investigation to future work. In more detail, our future work plans to exploit FastLog to learn logical rules with entity constants (Meilicke et al., 2024) and logical rules with type constraints (Wu et al., 2022).

8 Ethics Statement

This work presents FastLog, a framework for efficient end-to-end rule learning. Our evaluations rely on publicly available datasets, such as Freebase and Wikidata, which are widely used in academic research and do not contain private or sensitive information. We ensure that FastLog operates fairly across diverse datasets and provides transparency on its limitations to avoid unintended bias. While FastLog aims to improve the scalability and efficiency of KG reasoning, we emphasize the need for responsible use, particularly in sensitive applications. We encourage continuous monitoring and human oversight when deploying FastLog-based systems to mitigate potential risks.

References

Tolu Alabi, Jeffrey D. Blanchard, Bradley Gordon, and Russel Steinbach. 2012. Fast k-selection algorithms for graphics processing units. *ACM J. Exp. Algorithmics*, 17(1).

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. 2007. Dbpedia: A nucleus for a web of open data. In *ISWC*, volume 4825, pages 722–735.

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *NIPS*, pages 2787–2795.

William W. Cohen, Fan Yang, and Kathryn Mazaitis. 2020. Tensorlog: A probabilistic database implemented using deep-learning infrastructure. *J. Artif. Intell. Res. (JAIR)*, 67:285–325.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2d knowledge graph embeddings. In *AAAI*, pages 1811–1818.

Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. 2015. Fast rule mining in ontological knowledge bases with AMIE+. *VLDB J.*, 24(6):707–730.

Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian M. Suchanek. 2013. AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*, pages 413–422.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Adrian Kochsiek and Rainer Gemulla. 2021. Parallel training of knowledge graph embedding models: A comparison of techniques. *VLDB*, 15(3):633–645.

Stanley Kok and Pedro M. Domingos. 2007. Statistical predicate invention. In *ICML*, volume 227, pages 433–440.

Xiao Liu, Shiyu Zhao, Kai Su, Yukuo Cen, Jiezhong Qiu, Mengdi Zhang, Wei Wu, Yuxiao Dong, and Jie Tang. 2022. Mask and reason: Pre-training knowledge graph transformers for complex logical queries. In *KDD*, pages 1120–1130.

Linhao Luo, Yuan-Fang Li, Reza Haf, and Shirui Pan. 2024. Reasoning on graphs: Faithful and interpretable large language model reasoning. In *ICLR*.

Xinze Lyu, Guangyao Li, Jiacheng Huang, and Wei Hu. 2020. Rule-guided graph neural networks for recommender systems. In *ISWC*, pages 384–401.

Christian Meilicke, Melisachew Wudage Chekol, Patrick Betz, Manuel Fink, and Heiner Stuckenschmidt. 2024. Anytime bottom-up rule learning for large-scale knowledge graph completion. *VLDB J.*, 33(1):131–161.

Arindam Mitra and Chitta Baral. 2016. Addressing a question answering challenge by combining statistical methods with inductive rule learning and reasoning. In *AAAI*, pages 2779–2785.

Stephen H. Muggleton and Luc De Raedt. 1994. Inductive logic programming: Theory and methods. *J. Log. Program.*, 19/20:629–679.

Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying large language models and knowledge graphs: A roadmap. *IEEE Trans. Knowl. Data Eng. (TKDE)*, 36(7):3580–3599.

Kunxun Qi, Jianfeng Du, and Hai Wan. 2023. Learning from both structural and textual knowledge for inductive knowledge graph completion. In *NeurIPS*.

Meng Qu, Junkun Chen, Louis-Pascal A. C. Xhonneux, Yoshua Bengio, and Jian Tang. 2021. Rnnlogic: Learning logic rules for reasoning on knowledge graphs. In *ICLR*.

Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. 2019. DRUM: end-to-end differentiable rule mining on knowledge graphs. In *NeurIPS*, pages 15321–15331.

Apoorv Saxena, Adrian Kochsiek, and Rainer Gemulla. 2022. Sequence-to-sequence knowledge graph completion and question answering. In *ACL*, pages 2814–2828.

Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *ESWC*, volume 10843, pages 593–607.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *WWW*, pages 697–706.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. Rotate: Knowledge graph embedding by relational rotation in complex space. In *ICLR*.

Komal K. Teru, Etienne G. Denis, and William L. Hamilton. 2020. Inductive relation prediction by subgraph reasoning. In *ICML*, volume 119, pages 9448–9457.

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

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *ICML*, volume 48, pages 2071–2080.

Denny Vrandecic and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85.

Kai Wang, Yuwei Xu, and Siqiang Luo. 2024a. TIGER: training inductive graph neural network for large-scale knowledge graph reasoning. *VLDB*, 17(10):2459–2472.

Xiaozhi Wang, Tianyu Gao, Zhaocheng Zhu, Zhengyan Zhang, Zhiyuan Liu, Juanzi Li, and Jian Tang. 2021. KEPLER: A unified model for knowledge embedding and pre-trained language representation. *Trans. Assoc. Comput. Linguistics (TACL)*, 9:176–194.

Xiaxia Wang, David Jaime Tena Cucala, Bernardo Cuenca Grau, and Ian Horrocks. 2024b. Faithful rule extraction for differentiable rule learning models. In *ICLR*.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, pages 1112–1119.

Hong Wu, Zhe Wang, Kewen Wang, and Yi-Dong Shen. 2022. Learning typed rules over knowledge graphs. In *KR*.

Chenyan Xiong, Russell Power, and Jamie Callan. 2017. Explicit semantic ranking for academic search via knowledge graph embedding. In *WWW*, pages 1271–1279.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding entities and relations for learning and inference in knowledge bases. In *ICLR*.

Fan Yang, Zhilin Yang, and William W. Cohen. 2017. Differentiable learning of logical rules for knowledge base reasoning. In *NIPS*, pages 2319–2328.

Rongzhen Ye, Tianqu Zhuang, Hai Wan, Jianfeng Du, Weilin Luo, and Pingjia Liang. 2023. A noise-tolerant differentiable learning approach for single occurrence regular expression with interleaving. In *AAAI*, pages 4809–4817.

Qiang Zeng, Jignesh M. Patel, and David Page. 2014. Quickfoil: Scalable inductive logic programming. *VLDB*, 8(3):197–208.

Jingrong Zhang, Akira Naruse, Xipeng Li, and Yong Wang. 2023. Parallel top-k algorithms on GPU: A comprehensive study and new methods. In *SC*, pages 76:1–76:13.

Yongqi Zhang and Quanming Yao. 2022. Knowledge graph reasoning with relational digraph. In *WWW*, pages 912–924.

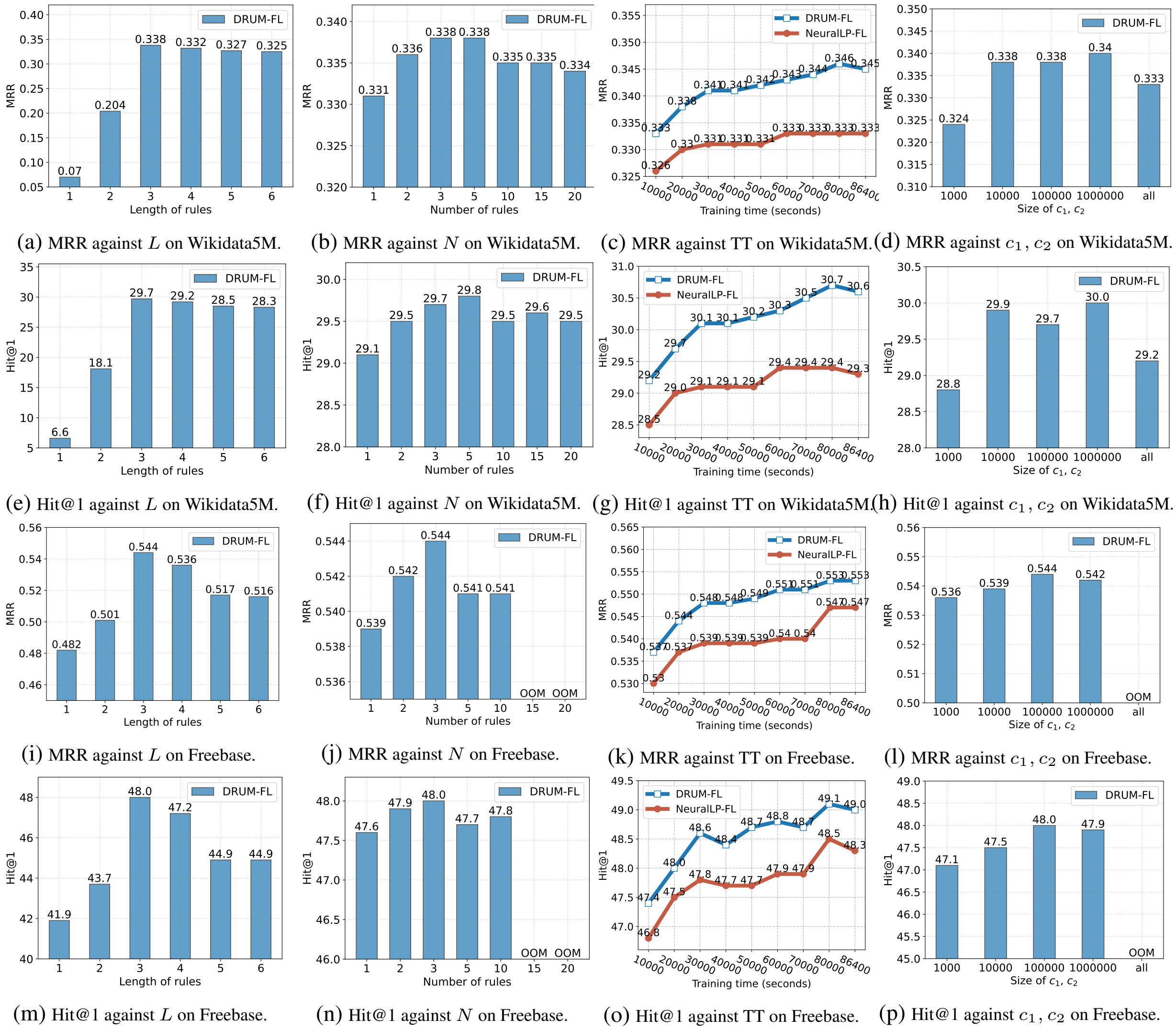
Zhaocheng Zhu, Xinyu Yuan, Michael Galkin, Louis-Pascal A. C. Xhonneux, Ming Zhang, Maxime Gazeau, and Jian Tang. 2023. A*net: A scalable path-based reasoning approach for knowledge graphs. In *NeurIPS*.

Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal A. C. Xhonneux, and Jian Tang. 2021. Neural bellman-ford networks: A general graph neural network framework for link prediction. In *NeurIPS*, pages 29476–29490.

A Empirical Analysis

A.1 Analysis on Learning Longer Rules

To verify whether the search-based method `AnyBURL` and the `FastLog`-enhanced methods can benefit from learning longer rules, we conducted an analysis on the efficacy of `DRUM-FL` using varying hyper-parameter L settings within a training time limit of 20,000 seconds, as reported in Table 7. We also created a variant (denoted by `AnyBURL`†) of `AnyBURL`by omitting the learning of logical rules with entity constants for a more comprehensive comparison. We can observe that both `AnyBURL` and `AnyBURL`† fail to be evaluated on



(a) MRR against L on Wikidata5M. (b) MRR against N on Wikidata5M. (c) MRR against TT on Wikidata5M. (d) MRR against c_1, c_2 on Wikidata5M.

(e) Hit@1 against L on Wikidata5M. (f) Hit@1 against N on Wikidata5M. (g) Hit@1 against TT on Wikidata5M. (h) Hit@1 against c_1, c_2 on Wikidata5M.

(i) MRR against L on Freebase. (j) MRR against N on Freebase. (k) MRR against TT on Freebase. (l) MRR against c_1, c_2 on Freebase.

(m) Hit@1 against L on Freebase. (n) Hit@1 against N on Freebase. (o) Hit@1 against TT on Freebase. (p) Hit@1 against c_1, c_2 on Freebase.

Figure 3: Analysis on different hyper-parameters

Wikidata5M within a reasonable time when longer rules ($L > 3$) were learnt. This can be attributed to the fact that reasoning with longer rules is time-consuming for search-based methods. In contrast, thanks to the highly parallel implementation of FastLog on a GPU, the FastLog-enhanced method DRUM-FL can be effectively evaluated on Wikidata5M within 1,000 seconds. In more detail, DRUM-FL achieves a 53.5x speedup over AnyBURL in evaluation efficiency when $L = 3$, with the speedup becoming even more evident as L increases. These results imply that the FastLog-enhanced methods are effective in learning longer rules. Besides, learning too many rules also impairs the explainability of AnyBURL.

The sub-figures (a) and (e) (resp. (i) and (m)) in Figure 3 illustrate the evaluation results of DRUM-FL using different L on the Wikidata5M (resp. Freebase) dataset. We found that DRUM-FL achieves the highest MRR and Hit@1 scores on both Wikidata5M and Freebase when $L = 3$, implying that DRUM-FL cannot benefit from learning longer rules. The reasons may be two-fold. Firstly, the training efficiency of DRUM-FL decreases as L increases, which in turn impairs the efficacy for link prediction under the same training time limit. Secondly, the search space increases exponentially with the increasing of L. This imposes a great challenge for end-to-end rule learning methods to learn precise rules on large-scale KGs, thereby leading to the decline of MRR and Hit@1 scores. Nevertheless, compared to search-based methods, the FastLog-enhanced methods are more effective in learning longer rules.

A.2 Analysis on Learning More Rules.

To verify whether the FastLog-enhanced methods can benefit from learning more rules, we conducted

Method	Wikidata5M						
	MRR	H@1	H@3	H@10	ECP	ET	NLR
AnyBURL($L=1$)	0.334	29.2	35.1	40.9	100%	5822s	7.0M
AnyBURL($L=2$)	0.351	30.8	36.8	42.6	100%	13940s	6.4M
AnyBURL($L=3$)	**0.355**	**31.3**	**37.2**	**43.2**	100%	18469s	6.2M
AnyBURL($L=4$)	-	-	-	-	47%	>1day	5.2M
AnyBURL($L=5$)	-	-	-	-	3.2%	>1day	4.7M
AnyBURL($L=6$)	-	-	-	-	0.5%	>1day	4.3M
AnyBURL† ($L=1$)	0.070	6.6	7.2	7.3	100%	1s	1K
AnyBURL† ($L=2$)	0.209	18.6	22.2	23.8	100%	237s	30K
AnyBURL† ($L=3$)	0.304	26.6	31.8	36.4	100%	20919s	97K
AnyBURL† ($L=4$)	-	-	-	-	23.9%	>1day	178K
AnyBURL† ($L=5$)	-	-	-	-	3.2%	>1day	202K
AnyBURL† ($L=6$)	-	-	-	-	0.7%	>1day	219K
DRUM-FL ($L=1$)	0.070	6.6	7.2	7.3	100%	68s	-
DRUM-FL ($L=2$)	0.204	18.1	21.8	23.7	100%	176s	-
DRUM-FL ($L=3$)	0.338	29.7	35.7	41.1	100%	341s	-
DRUM-FL ($L=4$)	0.332	29.2	35.0	40.8	100%	544s	-
DRUM-FL ($L=5$)	0.327	28.5	34.7	40.4	100%	731s	-
DRUM-FL ($L=6$)	0.325	28.3	34.6	40.4	100%	931s	-

Table 7: Comparison results against different L for AnyBURL and DRUM-FL on Wikidata5M, where ECP (resp. ET or NLR) abbreviates the evaluation completion progress (resp. evaluation time or the number of learnt rules).

an analysis on hyper-parameter N. The sub-figures (a) and (e) (resp. (i) and (m)) in Figure 3 illustrate the evaluation results of DRUM-FL using different N on the Wikidata5M (resp. Freebase) dataset. It can be seen that DRUM-FL is able to achieve higher MRR and Hit@1 scores as N increases when $1 \leq N \leq 5$. However, further increasing N does not lead to higher MRR and Hit@1 scores. This may be due to the fact that learning more rules may hinder the training efficiency of DRUM-FL, making some training cases ignored within the training time limit.

A.3 Analysis on More Training Time

To verify whether the FastLog-enhanced methods can benefit from more training time, we conducted an analysis on NeuralLP-FL and DRUM-FL with increasing training time, as illustrated in the sub-figures (c) and (g) (resp. (k) and (o)) in Figure 3 for the Wikidata5M (resp. Freebase) dataset. It can be seen that both NeuralLP-FL and DRUM-FL exhibit higher MRR and Hit@1 scores as more training time is allowed. Besides, we found that DRUM-FL is able to outperform AnyBURL(w/o constants) when the training time is not less than 20,000 seconds on Freebase. Note that AnyBURL cannot obtain efficacy improvement by allowing more training time (Meilicke et al., 2024). In contrast, the FastLog-enhanced methods can achieve better efficacy for link prediction as training time increases. These results suggest that we can further improve the efficacy of the FastLog-enhanced methods by allowing a longer training period.

Method	Wikidata5M				
	MRR	H@1	H@3	H@10	TT
DRUM-FL ($c_1, c_2 = 1,000$)	0.320	28.5	33.8	38.4	**11281s**
DRUM-FL ($c_1, c_2 = 10,000$)	0.331	28.9	35.3	40.6	12482s
DRUM-FL ($c_1, c_2 = 100,000$)	0.335	29.5	35.5	41.0	13473s
DRUM-FL ($c_1, c_2 = 1,000,000$)	**0.336**	**29.7**	35.4	40.8	18117s
DRUM-FL (w/o PS)	**0.336**	29.6	**35.6**	**41.1**	35303s

Table 8: Comparison results against different settings of c_1, c_2 for DRUM-FL.

A.4 Analysis on c_1 and c_2

To clarify why the proposed dynamic pruning strategy can improve efficacy, we conducted an analysis on DRUM-FL using varying settings for c_1 and c_2 within a training time limit of 20,000 seconds. The sub-figure (d) and (h) (resp. (l) and (p)) in Figure 3 illustrates the evaluation results of DRUM-FL with different setting of c_1 and c_2 on the Wikidata5M (resp. Freebase) dataset. We can see that DRUM-FL achieves higher MRR and Hit@1 scores as both c_1 and c_2 increases to 100,000. We can also see that the MRR and Hit@1 scores drop as either c_1 or c_2 further increases. To clarify why this happens, we further analyzed the efficacy of DRUM-FL using varying settings of c_1 and c_2 without limiting the training time, as reported by Table 8. It can be seen that the Hit@1 scores for DRUM-FL increase as c_1 and c_2 increase. We can also observe that DRUM-FL (w/o PS) is able to achieve the same MRR score as DRUM-FL ($c_1, c_2 = 1,000,000$) but spends much more training time. These results reveal that the proposed dynamic pruning strategy improves the efficacy for link prediction within a training time limit by significantly improving the training efficiency. Besides, we also analyzed the impacts of different combinations of c_1 and c_2, as reported in Table 9. Results suggest that the minimal combination to maximize the Hit@10 score is $c_1 = 100,000$ and $c_2 = 100,000$. Therefore, we recommend this setting as the default setting of FastLog.

B Proofs

In this section, we provide detailed proofs for all propositions in this work.

B.1 Proof of Proposition 1

Proof. (I) We first prove that the time complexity of a forward computation step for TensorLog is

c_1	c_2				
	1,000	10,000	100,000	1,000,000	$\lvert\mathcal{K}\rvert$
1,000	37.5	39.6	39.7	39.6	39.5
10,000	37.8	39.5	39.9	39.9	39.8
100,000	38.0	39.3	**40.0**	39.9	39.9
1,000,000	37.4	39.4	39.7	39.6	39.6
$\lvert\mathcal{E}\rvert$	37.5	39.4	**40.0**	39.5	39.5

Table 9: Hit@10 scores against different combinations of c_1 and c_2 for DRUM-FL on the validation set of Wikidata5M.

$\mathcal{O}(NL(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|))$. Let $\text{nnz}(M_{r_i})$ be the number of non-zero elements in the sparse matrix M_{r_i}. From Equations (1-2), we know that the complexity for each step in TensorLog comes from $\sum_{i=1}^{2n+1} \phi_{r,x}^{(k,l-1)}(w_i^{(r,k,l)} M_{r_i})$. Since the time complexity of $\sum_{i=1}^{2n+1} \phi_{r,x}^{(k,l-1)}(w_i^{(r,k,l)} M_{r_i})$ is $\sum_{i=1}^{2n+1}(\text{nnz}(M_{r_i}) + |\mathcal{E}|)$, where $n = |\mathcal{R}|$. By $\sum_{i=1}^{2n+1} \text{nnz}(M_{r_i}) = |\mathcal{K}|$, we can infer that the time complexity of a forward computation step for TensorLog is $\mathcal{O}(NL(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|))$.

(II) We then prove that the time complexity of a backward propagation step for TensorLog is $\mathcal{O}(NL(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|))$. For a backward propagation step, we know that only $w^{(r,k,l)}$ is trainable. The time complexity for calculating $\frac{\partial \mathcal{L}}{\partial \phi_{r,x}^{(k,l)}} M_{r_i}^{\top}$ is $\text{nnz}(M_{r_i}) + |\mathcal{E}|$. Therefore, the time complexity of a backward propagation step for TensorLog is $\mathcal{O}(NL\sum_{i=1}^{2n+1}(\text{nnz}(M_{r_i}) + |\mathcal{E}|)) = \mathcal{O}(NL(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|))$. □

B.2 Proof of Proposition 2

Proof. (I) From Equations (3-6), the time complexity of a forward computation step for FastLog is

$$NL(\underbrace{|\mathcal{K}|}_{\odot} + \underbrace{|\mathcal{K}|}_{\mathcal{F}_{\text{f2e}}})$$

Therefore, the time complexity of a forward computation step for FastLogis $\mathcal{O}(NL|\mathcal{K}|)$.

(II) For a backward propagation step of FastLog, we know that only $w^{(r,k,l)}$ is trainable. Let $z = \mathcal{F}_{\text{e2f}}(\phi_{r,x}^{(k,l-1)}) \odot \mathcal{F}_{\text{r2f}}(w^{(r,k,l)})$. The time complexity for calculating $\frac{\partial \mathcal{F}_{\text{f2e}}(z)}{\partial z}$ is $\mathcal{O}(|\mathcal{K}|)$. The time complexity for calculating $\frac{\partial z}{\partial \mathcal{F}_{\text{e2f}}(\phi_{r,x}^{(k,l-1)})}$ is $\mathcal{O}(|\mathcal{K}|)$. The time complexity for calculating $\frac{\partial z}{\partial \mathcal{F}_{\text{r2f}}(w^{(r,k,l)})}$ is $\mathcal{O}(|\mathcal{K}|)$. The time complexity for calculating $\frac{\partial \mathcal{F}_{\text{r2f}}(w^{(r,k,l)})}{\partial w^{(r,k,l)}}$ is $\mathcal{O}(|\mathcal{K}|)$. Therefore, the time complexity of a backward propagation step for FastLogis $\mathcal{O}(NL|\mathcal{K}|)$. □

B.3 Proof of Proposition 3

Proof. To prove Proposition 3, we first introduce three sparse matrices M_{e2f}, M_{r2f}, and M_{f2e}, where $M_{\text{e2f}} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{K}|}$ (resp. $M_{\text{r2f}} \in \mathbb{R}^{(2n+1)\times|\mathcal{K}|}$ or $M_{\text{f2e}} \in \mathbb{R}^{|\mathcal{K}|\times|\mathcal{E}|}$) stores the mapping between a head entity (resp. relation or fact) and its corresponding fact (resp. fact or tail entity).

For all $1 \leq k \leq N, 1 \leq l \leq L$, it holds that

$$\begin{aligned} \phi_{r,a}^{(k,l)} &= \mathcal{F}_{\text{f2e}}(\mathcal{F}_{\text{e2f}}(\phi_{r,a}^{(k,l-1)}) \odot \mathcal{F}_{\text{r2f}}(w^{(r,k,l)})) \\ &= ((\phi_{r,a}^{(k,l-1)} M_{\text{e2f}}) \odot (w^{(r,k,l)} M_{\text{r2f}}))M_{\text{f2e}} \\ &= \phi_{r,a}^{(k,l-1)}((M_{\text{e2f}} \odot (w^{(r,k,l)} M_{\text{r2f}}))M_{\text{f2e}}) \\ &= \phi_{r,a}^{(k,l-1)}(\sum_{i=1}^{2n+1} w_i^{(r,k,l)} M_{r_i}) \end{aligned}$$

Therefore, we have

$$\begin{aligned} \texttt{FastLog}(\theta_r^{N,L}, a, b) &= (\sum_{k=1}^{N} \phi_{r,a}^{(k,L)})v_b \\ &= (\sum_{k=1}^{N}((\cdots(v_a^{\top}(\sum_{i=1}^{2n+1} w_i^{(r,k,1)} M_{r_i})) \\ &\quad (\sum_{i=1}^{2n+1} w_i^{(r,k,2)} M_{r_i})) \\ &\quad \cdots \\ &\quad (\sum_{i=1}^{2n+1} w_i^{(r,k,L)} M_{r_i})))v_b \\ &= v_a^{\top}(\sum_{k=1}^{N}\prod_{l=1}^{L}\sum_{i=1}^{2n+1} w_i^{(r,k,l)} M_{r_i})v_b \\ &= \texttt{TensorLog}(\theta_r^{N,L}, a, b) \end{aligned}$$

□

B.4 Proof of Proposition 4

Proof. (I) We first prove that the space complexity of a forward computation step for TensorLog is $\mathcal{O}(m|\mathcal{E}|)$. For all $1 \leq k \leq N, 1 \leq l \leq L$, TensorLog requires a space of $m|\mathcal{E}|$ to store the intermediate estimated truth degrees. Since the summation of predicate selection is serial, this process does not require additional space. Therefore, the space complexity of a forward computation step for TensorLog is $\mathcal{O}(m|\mathcal{E}|)$.

(II) We then prove that the space complexity of a backward propagation step for TensorLog is $\mathcal{O}(mNL|\mathcal{R}||\mathcal{E}|)$. For a backward propagation step, TensorLog requires to store all intermediate estimated truth degrees for all L steps for all N rules to calculate the gradient. Therefore, the space complexity of a backward propagation step for TensorLog is $\mathcal{O}(mNL|\mathcal{R}||\mathcal{E}|)$. □

B.5 Proof of Proposition 5

Proof. (I) We first prove that the space complexity of a forward computation step for `FastLog` is $\mathcal{O}(m|\mathcal{K}|)$. For all $1 \leq k \leq N, 1 \leq l \leq L$, `FastLog` requires a space of the size $m|\mathcal{K}|$ to store the intermediate hidden state for all facts. Although `FastLog` also requires a space of $m|\mathcal{E}|$ to store the intermediate estimated truth degrees, it can reuse the previously opened space. In general, it holds that $|\mathcal{K}| > |\mathcal{E}|$. Therefore, a forward computation step for `FastLog` is $\mathcal{O}(m|\mathcal{K}|)$.

(II) We then prove that the space complexity of a backward propagation step for `FastLog` is $\mathcal{O}(mNL(|\mathcal{K}| + |\mathcal{E}|))$. For a backward propagation step, `FastLog` requires storing the intermediate hidden states for all L steps for all N rules to calculate the gradients. It also requires storing the intermediate estimated truth degrees for all L steps for all N rules to calculate the gradients. Therefore, the space complexity of a backward propagation step for `TensorLog` is $\mathcal{O}(mNL(|\mathcal{K}| + |\mathcal{E}|))$. □

B.6 Proof of Proposition 6

Proof. (I) We first prove that the time complexity of a forward computation step for $\texttt{FastLog}^{c_1,c_2}$ is $\mathcal{O}(NL(|\mathcal{K}| + |\mathcal{E}|))$. From Proposition 2, we know that the time complexity of a forward computation step for `FastLog` is $\mathcal{O}(NL|\mathcal{K}|)$. From Equation (8), we know that the dynamic pruning strategy introduces an additional complexity of $\mathcal{O}(NL|\mathcal{E}|)$ to calculate top-c_1 intermediate estimated truth degrees. From Equation (9), we know that the dynamic pruning strategy introduces an additional complexity of $\mathcal{O}(NL|\mathcal{K}|)$ to calculate top-c_2 intermediate hidden states. Therefore, the time complexity of a forward computation step for $\texttt{FastLog}^{c_1,c_2}$ is $\mathcal{O}(NL(|\mathcal{K}| + |\mathcal{E}|))$.

(II) We then prove that the time complexity of a backward propagation step for `FastLog` is $\mathcal{O}(NLc_2)$. From Equations (8), we know that only top-c_1 intermediate estimated truth degrees are used to calculate the gradients. From Equations (9), we know that only top-c_2 intermediate hidden states are used to calculate the gradients. Let $z = \hat{\mathcal{F}}^{c_2}_{\text{r2f}}(\hat{\mathcal{F}}^{c_1}_{\text{e2f}}(\phi^{(k,l-1)}_{r,x}), w^{(r,k,l)})$. The time complexity for calculating $\frac{\partial \hat{\mathcal{F}}_{\text{f2e}}(z)}{\partial z}$ is $\mathcal{O}(c_2)$ because z only has c_2 elements. The time complexity for calculating $\frac{\partial z}{\partial w^{(r,k,l)}}$ is $\mathcal{O}(c_2)$ because only the top-c_2 elements in $\hat{\mathcal{F}}^{c_1}_{\text{e2f}}(\phi^{(k,l-1)}_{r,x})$ are used to calculate gradients. The time complexity for calculating $\frac{\partial \hat{\mathcal{F}}^{c_1}_{\text{e2f}}(\phi^{(k,l-1)}_{r,x})}{\partial \phi^{(k,l-1)}_{r,x}}$ is $\mathcal{O}(c_2)$ because only the top-c_2 elements in $\hat{\mathcal{F}}^{c_1}_{\text{e2f}}(\phi^{(k,l-1)}_{r,x})$ are used to calculate gradients. Therefore, the time complexity of a backward propagation step for `FastLog` is $\mathcal{O}(NLc_2)$. □

B.7 Proof of Proposition 7

Proof. (I) We first prove that the space complexity of a forward computation step for `FastLog` is $\mathcal{O}(m|\mathcal{K}|)$. For all $1 \leq k \leq N, 1 \leq l \leq L$, `FastLog` requires a space of the size $m|\mathcal{K}|$ to store the intermediate hidden state for all facts. Although `FastLog` also requires a space of $m|\mathcal{E}|$ to store the intermediate estimated truth degrees, it can reuse the previously opened space. In general, it holds that $|\mathcal{K}| > |\mathcal{E}|$. Therefore, a forward computation step for `FastLog` is $\mathcal{O}(m|\mathcal{K}|)$.

(II) We then prove that the space complexity of a backward propagation step for `FastLog` is $\mathcal{O}(mNL(c_1 + c_2))$. For a backward propagation step, `FastLog` requires storing the intermediate estimated truth degrees with the size of c_1 for all L steps for all N rules to calculate the gradients. It also requires storing the intermediate hidden states with the size of c_2 for all L steps for all N rules to calculate the gradients. Therefore, the space complexity of a backward propagation step for `FastLog` is $\mathcal{O}(mNL(c_1 + c_2))$. □

B.8 Proof of Proposition 8

Proof. From Equations (3-5) and (8-10), we know that $\hat{\mathcal{F}}^{|\mathcal{E}|}_{\text{e2f}}$ (resp. $\hat{\mathcal{F}}^{|\mathcal{K}|}_{\text{r2f}}$ or $\hat{\mathcal{F}}_{\text{f2e}}$) is equivalent to $\mathcal{F}_{\text{e2f}}$ (resp. $\mathcal{F}_{\text{r2f}}$ or $\mathcal{F}_{\text{f2e}}$) because both $\mathcal{T}^{|\mathcal{E}|}(\mathbb{T})$ and $\mathcal{T}^{|\mathcal{K}|}(\mathbb{T})$ return the original set $\mathbb{T}$ of tuples. Therefore, Equation (6) can be derived by:

$$\begin{aligned}
\texttt{FastLog}(\theta^{N,L}_r, a, b) &= \sum_{k=1}^{N} \phi^{(k,L)}_{r,a} v_b \\
&= \sum_{k=1}^{N} \mathcal{F}_{\text{f2e}}(\mathcal{F}_{\text{r2f}}(w^{(r,k,L)}) \odot \mathcal{F}_{\text{e2f}}(\\
&\quad \cdots \\
&\quad \mathcal{F}_{\text{f2e}}(\mathcal{F}_{\text{r2f}}(w^{(r,k,2)}) \odot \mathcal{F}_{\text{e2f}}(\\
&\quad \mathcal{F}_{\text{f2e}}(\mathcal{F}_{\text{r2f}}(w^{(r,k,1)}) \odot \mathcal{F}_{\text{e2f}}(v_x^\top)))) \\
&\quad \cdots))v_b \\
&= \sum_{k=1}^{N} \hat{\mathcal{F}}_{\text{f2e}}(\hat{\mathcal{F}}^{|\mathcal{K}|}_{\text{r2f}}(\hat{\mathcal{F}}^{|\mathcal{E}|}_{\text{e2f}}(\\
&\quad \cdots \\
&\quad \hat{\mathcal{F}}_{\text{f2e}}(\hat{\mathcal{F}}^{|\mathcal{K}|}_{\text{r2f}}(\hat{\mathcal{F}}^{|\mathcal{E}|}_{\text{e2f}}(v_x^\top), w^{(r,k,1)})), \cdots), \\
&\quad w^{(r,k,L)})) \\
&= \texttt{FastLog}^{|\mathcal{E}|,|\mathcal{K}|}(\theta^{N,L}_r, a, b)
\end{aligned}$$

□

C Formalization of Existing Methods

In the following, we introduce four SOTA end-to-end rule learning methods that employ `TensorLog` to learn CRs, including `NeuralLP`, `DRUM`, `smDRUM`, `mmDRUM`.

C.1 The `NeuralLP` Method

`NeuralLP` (Yang et al., 2017) is the first work that exploits `TensorLog` operators to learn CRs. Specifically, `NeuralLP` introduces a set of additional learnable parameters to pay attention to previous steps, thereby learning CRs with dynamic length without using the identity relation. Besides, `NeuralLP` leverages LSTM (Hochreiter and Schmidhuber, 1997) networks to estimate both the predicate selection weights and the attention weights. Note that `NeuralLP` only simulates the inference of one CR, i.e., it holds that $N = 1$. Formally, given a query $(x, r, ?)$ and the maximum length of each rule L, `NeuralLP` first encodes r as a trainable vector $v_r \in \mathbb{R}^d$, where d denotes the dimensional size. Then an input sequence $(q_1, q_2, \cdots, q_{L+1})$ is created by setting $q_l = v_r$ for all $1 \leq l \leq L$ and $q_{L+1} = v^{\text{end}}$, where v^{end} is a special trainable vector to capture the boundary of the input sequence. For all $1 \leq l \leq L+1$, the predicate selection weights $w^{(r,1,l)} \in [0,1]^{2n}$ are estimated by

$$\begin{aligned} h_l &= \text{LSTM}(h_{l-1}, q_l), \\ w^{(r,1,l)} &= \text{Softmax}(W h_l + b), \end{aligned} \tag{13}$$

where h_0 is a zero-padding d-dimensional vector. $W \in \mathbb{R}^{2n \times d}$ and $b \in \mathbb{R}^{2n}$ are trainable weights and bias, respectively. The attention weights $\alpha^{(r,1,l)} \in [0,1]^l$ are estimated by

$$\alpha^{(r,1,l)} = \text{Softmax}([h_0^\top h_l; h_1^\top h_l; \cdots; h_{l-1}^\top h_l]), \tag{14}$$

where $[;]$ is the concatenation operation. The intermediate truth degrees $\phi_{r,x}^{(1,l)} \in \mathbb{R}^{|\mathcal{E}|}$ for `NeuralLP` are estimated by

$$\phi_{r,x}^{(1,l)} = \begin{cases} \sum_{i=1}^{2n} (\sum_{j=0}^{l-1} \alpha_j^{(r,1,l)} \phi_{r,x}^{(1,j)})(w_i^{(r,1,l)} M_{r_i}), & 1 \leq l \leq L, \\ \sum_{j=0}^{L} \alpha_j^{(r,1,L+1)} \phi_{r,x}^{(1,j)}, & l = L+1, \end{cases} \tag{15}$$

where $\phi_{r,x}^{(1,0)} = v_x^\top$. For an arbitrary triple $(x, r, y) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, the truth degree of (x, r, y) is estimated by

$$\texttt{NeuralLP}(\theta_r^{1,L}, x, y) = \phi_{r,x}^{(1,L+1)} v_y, \tag{16}$$

where $\theta_r^{1,L} = \{v_r, W, b\} \cup \theta_{\text{LSTM}}$ is a set of trainable parameters for the head relation r, and θ_{LSTM} is the set of parameters used in the LSTM network.

The following Proposition 9 shows the time complexity of `NeuralLP` .

Proposition 9. *Let $\mathcal{K} = \mathcal{G} \cup \mathcal{G}^-$. The time complexity of a forward computation step for* `NeuralLP` *is $\mathcal{O}(L(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|) + L^2|\mathcal{E}|)$. The time complexity of a backward propagation step for* `NeuralLP` *is $\mathcal{O}(L(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|) + L^2|\mathcal{E}|)$.*

Proof. (I) We first prove that the time complexity of a forward computation step for `NeuralLP` is $\mathcal{O}(L(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|) + L^2|\mathcal{E}|)$. From Equations (15-16), we know that the time complexity of a forward computation step for `NeuralLP` is

$$\underbrace{L(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|)}_{\texttt{TensorLog}} + \underbrace{\frac{L(L-1)}{2}|\mathcal{E}|}_{\text{Aggregation}} + \underbrace{(L+1)(8d^2)}_{\text{LSTM network}} + \underbrace{(d^2+d)}_{\text{MLP}}$$

where d denotes the hidden size. In general, it holds that $L(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|) + \frac{L(L-1)}{2}|\mathcal{E}| \gg (L+1)(8d^2) + (d^2+d)$. Therefore, the time complexity of a forward computation step for `NeuralLP` is $\mathcal{O}(L(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|) + L^2|\mathcal{E}|)$.

(II) We then prove that the time complexity of a forward computation step for `NeuralLP` is $\mathcal{O}(L(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|) + L^2|\mathcal{E}|)$. From Equations (13-16), we know that the time complexity of a backward propagation step for `NeuralLP` is

$$\underbrace{2L(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|)}_{\texttt{TensorLog}} + \underbrace{L(L-1)|\mathcal{E}|}_{\text{Aggregation}} + \underbrace{2(L+1)(8d^2)}_{\text{LSTM network}} + \underbrace{2(d^2+d)}_{\text{MLP}}$$

In general, it holds that $L(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|) + L(L-1)|\mathcal{E}| \gg +2(L+1)(8d^2) + 2(d^2+d)$. Therefore, the time complexity of a forward computation step for `NeuralLP` is $\mathcal{O}(L(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|) + L^2|\mathcal{E}|)$. □

C.2 The `DRUM` Method

Different from `NeuralLP`, `DRUM` (Sadeghian et al., 2019) introduces more trainable parameters to learn more CRs, and uses the identity relation to learn rules with dynamic length. Specifically, `DRUM` leverages N BiLSTM networks to estimate the predicate selection weights. Formally, given a query $(x, r, ?)$, the maximum number of rules to be learnt N, the maximum length of each rule L, `DRUM` first encodes r as a trainable vector $v_r \in \mathbb{R}^d$, where d denotes the dimensional size. For all $1 \leq k \leq N, 1 \leq l \leq L$, the predicate selection weights $w^{(r,k,l)} \in [0,1]^{2n+1}$

is estimated by

$$\begin{aligned}\overrightarrow{h}^{(k)}_{l} &= \overrightarrow{\text{BiLSTM}}^{(k)}(\overrightarrow{h}^{(k)}_{l-1}, v_r),\\ \overleftarrow{h}^{(k)}_{L-l+1} &= \overleftarrow{\text{BiLSTM}}^{(k)}(\overleftarrow{h}^{(k)}_{L-l}, v_r),\\ w^{(r,k,l)} &= \text{Softmax}(W[\overrightarrow{h}^{(k)}_{l}; \overleftarrow{h}^{(k)}_{L-l+1}] + b),\end{aligned} \tag{17}$$

where both $\overrightarrow{h}^{(k)}_{0}$ and $\overleftarrow{h}^{(k)}_{L+1}$ are set as zero-padding d-dimensional vectors. $W \in \mathbb{R}^{(2n+1)\times 2d}$ and $b \in \mathbb{R}^{2n+1}$ are trainable weights and bias, respectively. For an arbitrary triple $(x, r, y) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, the intermediate truth degrees $\phi^{(k,l)}_{r,x} \in \mathbb{R}^{|\mathcal{E}|}$ for DRUM are estimated by

$$\phi^{(k,l)}_{r,x} = \sum_{i=1}^{2n+1} \phi^{(k,l-1)}_{r,x}(w^{(r,k,l)}_i M_{r_i}), \tag{18}$$

where $\phi^{(k,0)}_{r,x} = v_x^\top$. The truth degree of (x, r, y) is estimated by

$$\text{DRUM}(\theta^{N,L}_r, x, y) = \sum_{k=1}^{N} \phi^{(k,L)}_{r,x} v_y, \tag{19}$$

where $\theta^{N,L}_r = \{v_r, W, b\} \cup \bigcup_{1\le k\le N} \theta^{(k)}_{\text{BiLSTM}}$ is a set of trainable parameters for the head relation r, and $\theta^{(k)}_{\text{BiLSTM}}$ is the set of parameters used in the k-th BiLSTM network.

C.3 The smDRUM Method

smDRUM (Wang et al., 2024b) is proposed to enhance the faithfulness between DRUM and CRs, by introducing new tensorized operations. Note that smDRUM uses the same way as DRUM to estimate the predicate selection weights. For an arbitrary triple $(x, r, y) \in \mathcal{E}\times\mathcal{R}\times\mathcal{E}$, the intermediate truth degrees $\phi^{(k,l)}_{r,x} \in \mathbb{R}^{|\mathcal{E}|}$ for smDRUM are estimated by

$$\phi^{(k,l)}_{r,x} = \sum_{i=1}^{2n+1} \phi^{(k,l-1)}_{r,x} \otimes (w^{(r,k,l)}_i M_{r_i}), \tag{20}$$

where $\phi^{(k,0)}_{r,x} = v_x^\top$, $\otimes$ is the *max-production* operator, i.e., given two matrices $U \in \mathbb{R}^{a\times m}$ and $V \in \mathbb{R}^{m\times b}$, $(U \otimes V)_{i,j} = \max_{k=1}^{m} U_{i,k} \cdot V_{k,j}$ for all $1 \le i \le a$ and $1 \le j \le b$. The truth degree of (x, r, y) is estimated by

$$\text{smDRUM}(\theta^{N,L}_r, x, y) = \sum_{k=1}^{N} \phi^{(k,L)}_{r,x} v_y, \tag{21}$$

where $\theta^{N,L}_r = \{v_r, W, b\} \cup \bigcup_{1\le k\le N} \theta^{(k)}_{\text{BiLSTM}}$ is a set of trainable parameters for the head relation r.

C.4 The mmDRUM Method

mmDRUM (Wang et al., 2024b) is another method proposed to enhance the faithfulness between DRUM and CRs. mmDRUM employs the same way in DRUM to estimate the predicate selection weights. Compared to smDRUM, mmDRUM introduces *max-pooling* to aggregate N rules. For an arbitrary triple $(x, r, y) \in \mathcal{E}\times\mathcal{R}\times\mathcal{E}$, the intermediate truth degrees $\phi^{(k,l)}_{r,x} \in \mathbb{R}^{|\mathcal{E}|}$ for mmDRUM are estimated by

$$\phi^{(k,l)}_{r,x} = \sum_{i=1}^{2n+1} \phi^{(k,l-1)}_{r,x} \otimes (w^{(r,k,l)}_i M_{r_i}), \tag{22}$$

where $\phi^{(k,0)}_{r,x} = v_x^\top$. The truth degree of (x, r, y) is estimated by

$$\text{mmDRUM}(\theta^{N,L}_r, x, y) = \max_{k=1}^{N} \phi^{(k,L)}_{r,x} v_y, \tag{23}$$

where $\theta^{N,L}_r = \{v_r, W, b\} \cup \bigcup_{1\le k\le N} \theta^{(k)}_{\text{BiLSTM}}$ is a set of trainable parameters for the head relation r.

The following Proposition 10 shows the time complexity of DRUM, smDRUM, and mmDRUM.

Proposition 10. *Let $\mathcal{K} = \mathcal{G} \cup \mathcal{G}^- \cup \{I(e, e) \mid e \in \mathcal{E}\}$. The time complexity of a forward computation step for* DRUM , smDRUM*, and* mmDRUM *is $\mathcal{O}(NL(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|))$. The time complexity of a backward propagation step for* DRUM , smDRUM*, and* mmDRUM *is $\mathcal{O}(NL(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|))$.*

Proof. From Equations (18-23), we know that DRUM, smDRUM and mmDRUM has the same training time complexity.

(I) We first prove that the time complexity of a forward computation step for DRUM, smDRUM, and mmDRUM is $\mathcal{O}(NL(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|))$. From Equations (17-19), we know that the time complexity of a forward computation step for DRUM is

$$\underbrace{NL(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|)}_{\text{TensorLog}} + \underbrace{2N(L+1)(8d^2)}_{\text{BiLSTM networks}} + \underbrace{N((2d)^2 + 2d)}_{\text{MLPs}}$$

where d denotes the hidden size. In general, it holds that $NL(|\mathcal{K}|+|\mathcal{R}||\mathcal{E}|) \gg N(L+1)(8d^2) + N((2d)^2 + 2d)$. Therefore, the time complexity of a forward computation step for DRUM, smDRUM, and mmDRUM is $\mathcal{O}(NL(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|))$.

(II) We then prove that the time complexity of a forward computation step for DRUM, smDRUM, and mmDRUM is $\mathcal{O}(NL(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|))$. From Equations (17-19), we know that the time complexity of a backward propagation step for DRUM, smDRUM, and mmDRUM is

$$\underbrace{2NL(|\mathcal{K}|+|\mathcal{R}||\mathcal{E}|)}_{\texttt{TensorLog}}+\underbrace{4N(L+1)(8d^2)}_{\text{BiLSTM networks}}+\underbrace{2N((2d)^2+2d)}_{\text{MLPs}}$$

In general, it holds that $2NL(|\mathcal{K}|+|\mathcal{R}||\mathcal{E}|) \gg 4N(L+1)(8d^2)+2N((2d)^2+2d)$. Therefore, the time complexity of a forward computation step for DRUM, smDRUM, and mmDRUM is $\mathcal{O}(NL(|\mathcal{K}|+|\mathcal{R}||\mathcal{E}|))$. □

C.5 Training objective

. The intuition of end-to-end rule learning methods is to search a set of parameters $\theta_r^{N,L}$ to distinguish positive facts from negative facts, by minimizing the following training objective.

$$\mathcal{L}(\{\theta_r^{N,L}\}_{r\in\mathcal{R}\cup\mathcal{R}^-}) = -\sum_{(x,r,y)\in\mathcal{G}\cup\mathcal{G}^-} \log \mathcal{M}(\theta_r^{N,L}, x, y), \tag{24}$$

where $\mathcal{M}$ is an end-to-end rule learning method or a FastLog-enhanced methods. Note that the loss is computed in a batch-wise parallel manner for all methods.

D FastLog-enhanced Methods

SOTA end-to-end rule learning methods can be enhanced by replacing TensorLog operators with FastLog operators. By X-FL we denote the FastLog-enhanced methods, where X can be NeuralLP, DRUM, smDRUM, and mmDRUM.

D.1 The NeuralLP-FL Method

In the following, we elaborate on enhancing the NeuralLP method with FastLog. It is worth noting that the use of FastLog does not affect the estimation of selection weights. Therefore, we can still use Equation (13-14) for selection weight estimation. Let $\mathcal{K}=\mathcal{G}\cup\mathcal{G}^-$. For an arbitrary triple $(x,r,y)\in\mathcal{E}\times\mathcal{R}\times\mathcal{E}$, the intermediate truth degrees $\phi_{r,x}^{(1,l)}\in\mathbb{R}^{|\mathcal{E}|}$ are estimated by

$$\phi_{r,x}^{(1,l)} = \begin{cases} \mathcal{F}_{\text{f2e}}(\mathcal{F}_{\text{e2f}}(\sum_{j=0}^{l-1}\alpha_j^{(r,1,l)}\phi_{r,x}^{(1,j)}) \odot \mathcal{F}_{\text{r2f}}(w^{(r,1,l)})), & 1\le l\le L,\\ \sum_{j=0}^{L}\alpha_j^{(r,1,L+1)}\phi_{r,x}^{(1,j)}, & l=L+1, \end{cases} \tag{25}$$

where $\phi_{r,x}^{(1,0)}=v_x$. The truth degree of (x,r,y) is estimated by

$$\texttt{NeuralLP}-\texttt{FL}(\theta_r^{1,L},x,y)=\phi_{r,x}^{(1,L+1)}v_y, \tag{26}$$

The following Proposition 11 shows the time complexity of NeuralLP-FL.

Proposition 11. *The time complexity of a forward computation step for* NeuralLP-FL *is* $\mathcal{O}(L|\mathcal{K}|+L^2|\mathcal{E}|)$. *The time complexity of a backward propagation step for* NeuralLP-FL *is* $\mathcal{O}(L|\mathcal{K}|+L^2|\mathcal{E}|)$.

Proof. (I) We first prove that the time complexity of a forward computation step for NeuralLP-FL is $\mathcal{O}(L|\mathcal{K}|+L^2|\mathcal{E}|)$. From Proposition 2, we know that the time complexity of a forward computation step for FastLogis $\mathcal{O}(NL|\mathcal{K}|)$. From Equations (25-26), we know that the time complexity of a forward computation step for NeuralLP-FL is

$$\underbrace{L|\mathcal{K}|}_{\texttt{FastLog}}+\underbrace{\frac{L(L-1)}{2}|\mathcal{E}|}_{\text{Aggregation}}+\underbrace{(L+1)(8d^2)}_{\text{LSTM network}}+\underbrace{(d^2+d)}_{\text{MLP}}$$

where d denotes the hidden size. In general, it holds that $L|\mathcal{K}|+\frac{L(L-1)}{2}|\mathcal{E}| \gg (L+1)(8d^2)+(d^2+d)$. Therefore, the time complexity of a forward computation step for NeuralLP-FL is $\mathcal{O}(L|\mathcal{K}|+L^2|\mathcal{E}|)$.

(II) We then prove that the time complexity of a backward propagation step for NeuralLP-FL is $\mathcal{O}(L|\mathcal{K}|+L^2|\mathcal{E}|)$. For a backward propagation step for NeuralLP-FL, we know that both $w^{(r,1,l)}$ and $\alpha^{(r,1,l)}$ are trainable. Let $z=\mathcal{F}_{\text{e2f}}(\sum_{j=0}^{l-1}\alpha_j^{(r,1,l)}\phi_{r,x}^{(1,j)})\odot\mathcal{F}_{\text{r2f}}(w^{(r,1,l)})$. The time complexity for calculating $\frac{\partial\mathcal{F}_{\text{f2e}}(z)}{\partial z}$ is $\mathcal{O}(|\mathcal{K}|)$. The time complexity for calculating $\frac{\partial z}{\partial\mathcal{F}_{\text{e2f}}(\sum_{j=0}^{l-1}\alpha_j^{(r,1,l)}\phi_{r,x}^{(1,j)})}$ is $\mathcal{O}(|\mathcal{K}|)$. The time complexity for calculating $\frac{\partial\mathcal{F}_{\text{e2f}}(\sum_{j=0}^{l-1}\alpha_j^{(r,1,l)}\phi_{r,x}^{(1,j)})}{\partial\alpha^{(r,1,l)}}$ is $\mathcal{O}(L^2|\mathcal{E}|)$. The time complexity for calculating $\frac{\partial z}{\partial\mathcal{F}_{\text{r2f}}(w^{(r,1,l)})}$ is $\mathcal{O}(|\mathcal{K}|)$. The time complexity for calculating $\frac{\partial\mathcal{F}_{\text{r2f}}(w^{(r,1,l)})}{\partial w^{(r,1,l)}}$ is $\mathcal{O}(|\mathcal{K}|)$. Therefore, the time complexity of a backward propagation step for NeuralLP-FL is $\mathcal{O}(L|\mathcal{K}|+L^2|\mathcal{E}|)$. □

By being enhanced by FastLog, the time complexity of a forward computation step for NeuralLP is reduced from $\mathcal{O}(L(|\mathcal{K}|+|\mathcal{R}||\mathcal{E}|)+L^2|\mathcal{E}|)$ to $\mathcal{O}(L|\mathcal{K}|+L^2|\mathcal{E}|)$. The time complexity of a backward propagation step for NeuralLP is reduced from $\mathcal{O}(L(|\mathcal{K}|+|\mathcal{R}||\mathcal{E}|)+L^2|\mathcal{E}|)$ to $\mathcal{O}(L|\mathcal{K}|+L^2|\mathcal{E}|)$. The following Proposition 12 demonstrates the correctness of NeuralLP-FL.

Proposition 12. *For an arbitrary triple* $(a,r,b)\in\mathcal{E}\times\mathcal{R}\times\mathcal{E}, \forall L\ge 1:$ $\texttt{NeuralLP}-\texttt{FL}(\theta_r^{1,L},a,b)=\texttt{NeuralLP}(\theta_r^{1,L},a,b)$.

Proof. To prove Proposition 12, we first introduce three sparse matrices M_{e2f}, M_{r2f}, and M_{f2e}, where $M_{\text{e2f}} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{K}|}$ (resp. $M_{\text{r2f}} \in \mathbb{R}^{2n\times|\mathcal{K}|}$ or $M_{\text{f2e}} \in \mathbb{R}^{|\mathcal{K}|\times|\mathcal{E}|}$) stores the mapping between a head entity (resp. relation or fact) and its corresponding fact (resp. fact or tail entity).

For all $1 \le l \le L$, it holds that

$$
\begin{aligned}
\phi_{r,a}^{(l)} &= \mathcal{F}_{\text{f2e}}(\mathcal{F}_{\text{e2f}}(\sum_{j=0}^{l-1}\alpha_j^{(r,l)}\phi_{r,a}^{(j)}) \odot \mathcal{F}_{\text{r2f}}(w^{(r,l)})) \\
&= ((\sum_{j=0}^{l-1}\alpha_j^{(r,l)}\phi_{r,a}^{(j)})M_{\text{e2f}} \odot (w^{(r,l)}M_{\text{r2f}}))M_{\text{f2e}} \\
&= (\sum_{j=0}^{l-1}\alpha_j^{(r,l)}\phi_{r,a}^{(j)})((M_{\text{e2f}} \odot (w^{(r,l)}M_{\text{r2f}}))M_{\text{f2e}}) \\
&= (\sum_{j=0}^{l-1}\alpha_j^{(r,l)}\phi_{r,a}^{(j)})(\sum_{i=1}^{2n} w_i^{(r,l)}M_{r_i}) \\
&= \sum_{i=1}^{2n}(\sum_{j=0}^{l-1}\alpha_j^{(r,l)}\phi_{r,a}^{(j)})(w_i^{(r,l)}M_{r_i})
\end{aligned}
$$

Therefore, we have

$$
\begin{aligned}
\texttt{NeuralLP}-\text{FL}(\theta_r^L, a, b) &= \phi_{r,a}^{(L+1)}v_b \\
&= \sum_{j=0}^{L}\alpha_j^{(r,L+1)}\phi_{r,x}^{(j)} \\
&= \sum_{j=0}^{L}\alpha_j^{(r,L+1)}(\\
&\quad \sum_{i=1}^{2n}(\sum_{k=0}^{j-1}\alpha_k^{(r,j)}\phi_{r,a}^{(k)})(w_i^{(r,j)}M_{r_i})) \\
&= \texttt{NeuralLP}(\theta_r^L, a, b)
\end{aligned}
$$

□

This proposition reveals that the efficacy of `NeuralLP` will not be impaired by applying `FastLog`.

D.2 The `DRUM-FL` Method

In the following, we elaborate on enhancing the `DRUM` method with `FastLog`. Let $\mathcal{K} = \mathcal{G} \cup \mathcal{G}^- \cup \{I(e,e) \mid e \in \mathcal{E}\}$. Similarly with `DRUM`, `DRUM-FL` also uses Equation (7) for selection weight estimation. For all $1 \le k \le N, 1 \le l \le L$, the intermediate truth degrees $\phi_{r,x}^{(k,l)} \in \mathbb{R}^{|\mathcal{E}|}$ are estimated by

$$
\phi_{r,x}^{(k,l)} = \mathcal{F}_{\text{f2e}}(\mathcal{F}_{\text{e2f}}(\phi_{r,x}^{(k,l-1)}) \odot \mathcal{F}_{\text{r2f}}(w^{(r,k,l)})), \tag{27}
$$

where $\phi_{r,x}^{(k,0)} = v_x^\top$. The truth degree of (x, r, y) is estimated by

$$
\texttt{DRUM}-\text{FL}(\theta_r^{N,L}, x, y) = (\sum_{k=1}^{N}\phi_{r,x}^{(k,L)})v_y, \tag{28}
$$

The following Proposition 13 shows the time complexity of `DRUM-FL`.

Proposition 13. *The time complexity of a forward computation step for* `DRUM-FL` *is* $\mathcal{O}(NL|\mathcal{K}|)$. *The time complexity of a backward propagation step for* `DRUM-FL` *is* $\mathcal{O}(NL|\mathcal{K}|)$.

Proof. (I) We first prove that the time complexity of a forward computation step for `DRUM-FL` is $\mathcal{O}(NL|\mathcal{K}|)$. From Proposition 2, we know that the time complexity of a forward computation step for `FastLog` is $\mathcal{O}(NL|\mathcal{K}|)$. From Equations (27-28), we know that the time complexity of a forward computation step for `DRUM-FL` is

$$
\underbrace{NL|\mathcal{K}|}_{\texttt{FastLog}} + \underbrace{2N(L+1)(8d^2)}_{\text{BiLSTM networks}} + \underbrace{N((2d)^2+2d)}_{\text{MLPs}}
$$

where d denotes the hidden size. In general, it holds that $NL|\mathcal{K}| \gg +2N(L+1)(8d^2)+((2d)^2+2d)$. Therefore, the time complexity of a forward computation step for `DRUM-FL` is $\mathcal{O}(NL|\mathcal{K}|)$.

(II) We then prove that the time complexity of a backward propagation step for `DRUM-FL` is $\mathcal{O}(NL|\mathcal{K}|)$. For a backward propagation step of `DRUM-FL`, we know that only $w^{(r,k,l)}$ is trainable. Let $z = \mathcal{F}_{\text{e2f}}(\phi_{r,x}^{(k,l-1)}) \odot \mathcal{F}_{\text{r2f}}(w^{(r,k,l)})$. The time complexity for calculating $\frac{\partial \mathcal{F}_{\text{f2e}}(z)}{\partial z}$ is $\mathcal{O}(|\mathcal{K}|)$. The time complexity for calculating $\frac{\partial z}{\partial \mathcal{F}_{\text{e2f}}(\phi_{r,x}^{(k,l-1)})}$ is $\mathcal{O}(|\mathcal{K}|)$. The time complexity for calculating $\frac{\partial z}{\partial \mathcal{F}_{\text{r2f}}(w^{(r,k,l)})}$ is $\mathcal{O}(|\mathcal{K}|)$. The time complexity for calculating $\frac{\partial \mathcal{F}_{\text{r2f}}(w^{(r,k,l)})}{\partial w^{(r,l,l)}}$ is $\mathcal{O}(|\mathcal{K}|)$. Therefore, the time complexity of a backward propagation step for `DRUM-FL` is $\mathcal{O}(NL|\mathcal{K}|)$. □

By being enhanced by `FastLog`, the time complexity of a forward computation step for `DRUM` is reduced from $\mathcal{O}(NL(|\mathcal{K}|+|\mathcal{R}||\mathcal{E}|))$ to $\mathcal{O}(NL|\mathcal{K}|)$. The time complexity of a backward propagation step for `DRUM` is reduced from $\mathcal{O}(NL(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|))$ to $\mathcal{O}(NL|\mathcal{K}|)$. The following Proposition 14 demonstrates the correctness of `DRUM-FL`.

Proposition 14. *For an arbitrary triple* $(a, r, b) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, $\forall N \ge 1, L \ge 1$: $\texttt{DRUM}-\text{FL}(\theta_r^{N,L}, a, b) = \texttt{DRUM}(\theta_r^{N,L}, a, b)$.

Proof. To prove Proposition 14, we first introduce three sparse matrices M_{e2f}, M_{r2f}, and M_{f2e}, where $M_{\mathrm{e2f}} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{K}|}$ (resp. $M_{\mathrm{r2f}} \in \mathbb{R}^{(2n+1)\times|\mathcal{K}|}$ or $M_{\mathrm{f2e}} \in \mathbb{R}^{|\mathcal{K}|\times|\mathcal{E}|}$) stores the mapping between a head entity (resp. relation or fact) and its corresponding fact (resp. fact or tail entity).

For all $1 \le k \le N, 1 \le l \le L$, it holds that

$$\begin{aligned}\phi_{r,a}^{(k,l)} &= \mathcal{F}_{\mathrm{f2e}}(\mathcal{F}_{\mathrm{e2f}}(\phi_{r,a}^{(k,l-1)}) \odot \mathcal{F}_{\mathrm{r2f}}(w^{(r,k,l)}))\\ &= ((\phi_{r,a}^{(k,l-1)} M_{\mathrm{e2f}}) \odot (w^{(r,k,l)} M_{\mathrm{r2f}}))M_{\mathrm{f2e}}\\ &= \phi_{r,a}^{(k,l-1)}((M_{\mathrm{e2f}} \odot (w^{(r,k,l)} M_{\mathrm{r2f}}))M_{\mathrm{f2e}})\\ &= \phi_{r,a}^{(k,l-1)}(\sum_{i=1}^{2n+1} w_i^{(r,k,l)} M_{r_i})\end{aligned}$$

Therefore, we have

$$\begin{aligned}\texttt{DRUM}\mathrm{-FL}(\theta_r^{N,L}, a, b) &= (\sum_{k=1}^{N} \phi_{r,a}^{(k,L)})v_b\\ &= (\sum_{k=1}^{N}((\cdots(v_a^\top(\sum_{i=1}^{2n+1} w_i^{(r,k,1)} M_{r_i}))\\ &(\sum_{i=1}^{2n+1} w_i^{(r,k,2)} M_{r_i}))\\ &\cdots\\ &(\sum_{i=1}^{2n+1} w_i^{(r,k,L)} M_{r_i})))v_b\\ &= v_a^\top(\sum_{k=1}^{N}\prod_{l=1}^{L}\sum_{i=1}^{2n+1} w_i^{(r,k,l)} M_{r_i})v_b\\ &= \texttt{DRUM}(\theta_r^{N,L}, a, b)\end{aligned}$$

□

This proposition reveals that the efficacy of `DRUM` will not be impaired by applying `FastLog`.

D.3 The `smDRUM`-FL Method

Similar to `DRUM`-FL, for all $1 \le k \le N, 1 \le l \le L$, the formalization of `smDRUM`-FL is defined as

$$\phi_{r,x}^{(k,l)} = \mathcal{F}_{\mathrm{f2e}}^{\max}(\mathcal{F}_{\mathrm{e2f}}(\phi_{r,x}^{(k,l-1)}) \odot \mathcal{F}_{\mathrm{r2f}}(w^{(r,k,l)})), \tag{29}$$

where $\phi_{r,x}^{(k,0)} = v_x^\top$. $\mathcal{F}_{\mathrm{f2e}}^{\max} : \mathbb{R}^{|\mathcal{K}|} \to \mathbb{R}^{|\mathcal{E}|}$ is a function such that the i-th elements of $\mathcal{F}_{\mathrm{f2e}}^{\max}(v)$ is

$$[\mathcal{F}_{\mathrm{f2e}}^{\max}(v)]_i = \max_{j:\mathrm{tail}(\tau_j)=i} v_j. \tag{30}$$

The truth degree of (x, r, y) is estimated by

$$\texttt{smDRUM}\mathrm{-FL}(\theta_r^{N,L}, x, y) = (\sum_{k=1}^{N} \phi_{r,x}^{(k,L)})v_y. \tag{31}$$

Note that `smDRUM`-FL has the same time complexity as `DRUM`-FL. The following Proposition 15 demonstrates the correctness of `smDRUM`-FL.

Proposition 15. *For an arbitrary triple* $(a, r, b) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, $\forall N \ge 1, L \ge 1$: $\texttt{smDRUM}\mathrm{-FL}(\theta_r^{N,L}, a, b) = \texttt{smDRUM}(\theta_r^{N,L}, a, b)$.

Proof. To prove Proposition 15, we first introduce three sparse matrices M_{e2f}, M_{r2f}, and M_{f2e}, where $M_{\mathrm{e2f}} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{K}|}$ (resp. $M_{\mathrm{r2f}} \in \mathbb{R}^{(2n+1)\times|\mathcal{K}|}$ or $M_{\mathrm{f2e}} \in \mathbb{R}^{|\mathcal{K}|\times|\mathcal{E}|}$) stores the mapping between a head entity (resp. relation or fact) and its corresponding fact (resp. fact or tail entity).

For all $1 \le k \le N, 1 \le l \le L$, it holds that

$$\begin{aligned}\phi_{r,a}^{(k,l)} &= \mathcal{F}_{\mathrm{f2e}}^{\max}(\mathcal{F}_{\mathrm{e2f}}(\phi_{r,a}^{(k,l-1)}) \odot \mathcal{F}_{\mathrm{r2f}}(w^{(r,k,l)}))\\ &= ((\phi_{r,a}^{(k,l-1)} M_{\mathrm{e2f}}) \odot (w^{(r,k,l)} M_{\mathrm{r2f}})) \otimes M_{\mathrm{f2e}}\\ &= \phi_{r,a}^{(k,l-1)}((M_{\mathrm{e2f}} \odot (w^{(r,k,l)} M_{\mathrm{r2f}})) \otimes M_{\mathrm{f2e}})\\ &= \phi_{r,a}^{(k,l-1)} \otimes ((M_{\mathrm{e2f}} \odot (w^{(r,k,l)} M_{\mathrm{r2f}}))M_{\mathrm{f2e}})\\ &= \phi_{r,a}^{(k,l-1)} \otimes (\sum_{i=1}^{2n+1} w_i^{(r,k,l)} M_{r_i})\end{aligned}$$

Therefore, we have

$$\begin{aligned}\texttt{smDRUM}\mathrm{-FL}(\theta_r^{N,L}, a, b) &= (\sum_{k=1}^{N} \phi_{r,a}^{(k,L)})v_b\\ &= (\sum_{k=1}^{N}((\cdots(v_a^\top \otimes (\sum_{i=1}^{2n+1} w_i^{(r,k,1)} M_{r_i}))\\ &\otimes(\sum_{i=1}^{2n+1} w_i^{(r,k,2)} M_{r_i}))\\ &\cdots\\ &\otimes(\sum_{i=1}^{2n+1} w_i^{(r,k,L)} M_{r_i})))v_b\\ &= v_a^\top(\sum_{k=1}^{N}\bigotimes_{l=1}^{L}\sum_{i=1}^{2n+1} w_i^{(r,k,l)} M_{r_i})v_b\\ &= \texttt{smDRUM}(\theta_r^{N,L}, a, b)\end{aligned}$$

□

This proposition reveals that the efficacy of `smDRUM` will not be impaired by applying `FastLog`.

D.4 The `mmDRUM`-FL Method

Similar to `DRUM`-FL and `smDRUM`-FL, the formalization of `mmDRUM`-FL is defined as

$$\phi_{r,x}^{(k,l)} = \mathcal{F}_{\mathrm{f2e}}^{\max}(\mathcal{F}_{\mathrm{e2f}}(\phi_{r,x}^{(k,l-1)}) \odot \mathcal{F}_{\mathrm{r2f}}(w^{(r,k,l)})), \tag{32}$$

where $\phi_{r,x}^{(k,0)} = v_x^\top$. The truth degree of (x, r, y) is estimated by

$$\texttt{mmDRUM}\mathrm{-FL}(\theta_r^{N,L}, x, y) = \max_{k=1}^{N} \phi_{r,x}^{(k,L)} v_y. \tag{33}$$

Note that `mmDRUM`-FL has the same time complexity as `DRUM`-FL and `smDRUM`-FL. The following Proposition 16 shows the correctness of `mmDRUM`-FL.

Proposition 16. *For an arbitrary triple* $(a, r, b) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, $\forall N \geq 1, L \geq 1$: $\texttt{mmDRUM}{-}\mathrm{FL}(\theta_r^{N,L}, a, b) = \texttt{mmDRUM}(\theta_r^{N,L}, a, b)$.

Proof. To prove Proposition 16, we first introduce three sparse matrices M_{e2f}, M_{r2f}, and M_{f2e}, where $M_{\mathrm{e2f}} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{K}|}$ (resp. $M_{\mathrm{r2f}} \in \mathbb{R}^{(2n+1)\times|\mathcal{K}|}$ or $M_{\mathrm{f2e}} \in \mathbb{R}^{|\mathcal{K}|\times|\mathcal{E}|}$) stores the mapping between a head entity (resp. relation or fact) and its corresponding fact (resp. fact or tail entity).

For all $1 \leq k \leq N, 1 \leq l \leq L$, it holds that

$$\begin{aligned}
\phi_{r,a}^{(k,l)} &= \mathcal{F}_{\mathrm{f2e}}^{\max}(\mathcal{F}_{\mathrm{e2f}}(\phi_{r,a}^{(k,l-1)}) \odot \mathcal{F}_{\mathrm{r2f}}(w^{(r,k,l)})) \\
&= ((\phi_{r,a}^{(k,l-1)} M_{\mathrm{e2f}}) \odot (w^{(r,k,l)} M_{\mathrm{r2f}})) \otimes M_{\mathrm{f2e}} \\
&= \phi_{r,a}^{(k,l-1)}((M_{\mathrm{e2f}} \odot (w^{(r,k,l)} M_{\mathrm{r2f}})) \otimes M_{\mathrm{f2e}}) \\
&= \phi_{r,a}^{(k,l-1)} \otimes ((M_{\mathrm{e2f}} \odot (w^{(r,k,l)} M_{\mathrm{r2f}}))M_{\mathrm{f2e}}) \\
&= \phi_{r,a}^{(k,l-1)} \otimes (\sum_{i=1}^{2n+1} w_i^{(r,k,l)} M_{r_i})
\end{aligned}$$

Therefore, we have

$$\begin{aligned}
\texttt{mmDRUM}{-}\mathrm{FL}(\theta_r^{N,L}, a, b) &= (\max_{k=1}^{N} \phi_{r,a}^{(k,L)}) v_b \\
&= (\max_{k=1}^{N}((\cdots(v_a^\top \otimes (\sum_{i=1}^{2n+1} w_i^{(r,k,1)} M_{r_i})) \\
&\quad \otimes (\sum_{i=1}^{2n+1} w_i^{(r,k,2)} M_{r_i})) \\
&\quad \cdots \\
&\quad \otimes (\sum_{i=1}^{2n+1} w_i^{(r,k,L)} M_{r_i}))) v_b \\
&= v_a^\top (\max_{k=1}^{N} \bigotimes_{l=1}^{L} \sum_{i=1}^{2n+1} w_i^{(r,k,l)} M_{r_i}) v_b \\
&= \texttt{mmDRUM}(\theta_r^{N,L}, a, b)
\end{aligned}$$

□

This proposition reveals that the efficacy of `mmDRUM` will not be impaired by applying `FastLog`.

E Discussion on Embedding-based Methods

Knowledge graph embeddings (KGEs) (Bordes et al., 2013; Wang et al., 2014; Yang et al., 2015; Trouillon et al., 2016; Sun et al., 2019) are a kind of typical methods for link prediction over KGs. They usually represent entities and relations in KGs as low-dimensional real-value vectors, and then estimate the truth degree of a triple based on the semantic distance or similarity calculated from entity and relation embeddings. However, KGE methods can hardly measure the triples involving previously unseen entities as their embeddings have not been trained. Besides, the learnt embeddings are real-value vectors that can hardly be interpreted. Adapting KGE methods to large KGs is non-trivial. Kochsiek and Gemulla (2021) employed 8 GPUs with a total of 88GB memory to train SOTA KGE methods on Wikidata5M and Freebase. In contrast, `FastLog` enables scalable end-to-end rule learning from large-scale KGs using a single GPU with 24 GB memory.

Graph neural networks (GNNs) (Schlichtkrull et al., 2018; Teru et al., 2020; Zhu et al., 2021; Zhang and Yao, 2022; Zhu et al., 2023) are a kind of embedding-based methods for link prediction. They can handle the inductive setting where missing triples involve unseen entities. However, GNN-based methods are still black-box methods that are difficult to interpret. In contrast, we focus on learning logical rules from large-scale KGs for better explainability. It is worth noting that `TIGER` (Wang et al., 2024a) employs a rapid sub-graph extraction algorithm to facilitate GNNs for link prediction over large-scale KGs. However, sub-graph extraction cannot take effect in reducing the time cost in some application scenarios where the given KG has no small sub-graphs for multi-hop reasoning. Therefore, we do not consider exploiting sub-graph extraction to enhance the efficiency of end-to-end rule learning.

More recently, there has been an increasing interest in leveraging pre-trained language models (PLMs) (Wang et al., 2021; Saxena et al., 2022; Liu et al., 2022) or even large language models (LLMs) (Luo et al., 2024; Pan et al., 2024) for link prediction over KGs. These methods are also embedding-based. They aim to leverage the pre-trained knowledge from text corpora and the contextual information of entities and relations to enhance the efficacy for link prediction. Based on the contextual information, PLM-based methods can handle previously unseen entities and relations. However, PLMs especially LLMs require massive computation resources such as GPU memory. Besides, they are black-box methods that lack interpretability. In contrast, the `FastLog`-enhanced methods have only moderate memory cost on GPUs, and they can interpret logical rules as explanations for missing triples.