Reconstructing TensorLog for Scalable End-to-end Rule Learning

Supplementary Materials

I. Proofs

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

A. Proof of Proposition 1

Proposition 1. *The time complexity for the training phase of* TensorLog *is* $\mathcal{O}(NL(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|))$.

Proof. (I) We first prove that the time complexity of a forward computation step for TensorLog is $\mathcal{O}(NL(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|))$. Let $\mathrm{nnz}(M_{r_i})$ 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 $\sum_{i=1}^{2n+1}(\mathrm{nnz}(M_{r_i}) + |\mathcal{E}|)$, where $n = |\mathcal{R}|$. By $\sum_{i=1}^{2n+1} \mathrm{nnz}(M_{r_i}) = |\mathcal{K}|$, 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 $\mathrm{nnz}(M_{r_i}) + |\mathcal{E}|$. Therefore, the time complexity of a backward propagation step for TensorLog is $\mathcal{O}(NL\sum_{i=1}^{2n+1}(\mathrm{nnz}(M_{r_i})+|\mathcal{E}|)) = \mathcal{O}(NL(|\mathcal{K}|+|\mathcal{R}||\mathcal{E}|))$. □

B. Proof of Proposition 2

Proposition 2. *The time complexity for the training phase of* FastLog *is* $\mathcal{O}(NL|\mathcal{K}|)$.

Proof. (I) From Equations (4-7), the time complexity of a forward computation step for FastLog is

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

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

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

C. Proof of Proposition 3

Proposition 3. *The space complexity for the training phase of* TensorLog *is* $\mathcal{O}(NL|\mathcal{R}||\mathcal{E}|)$.

Proof. (I) We first prove that the space complexity of a forward computation step for TensorLog is $\mathcal{O}(NL|\mathcal{R}||\mathcal{E}|)$. For a forward computation step, TensorLog requires storing all intermediate estimated truth degrees for all L steps for all N rules to calculate the gradient. Therefore, the space complexity of a forward computation step for TensorLog is $\mathcal{O}(NL|\mathcal{R}||\mathcal{E}|)$.

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

D. Proof of Proposition 4

Proposition 4. *The space complexity for the training phase of* FastLog *is* $\mathcal{O}(NL(|\mathcal{R}| + |\mathcal{E}| + |\mathcal{K}|))$.

Proof. (I) We first prove that the space complexity of a forward computation step for FastLog is $\mathcal{O}(NL(|\mathcal{R}|+|\mathcal{E}|+|\mathcal{K}|))$.For a forward computation step, FastLog requires storing the intermediate hidden states for all L steps for all N rules to calculate the gradients, resulting in a space complexity of $\mathcal{O}(NL|\mathcal{K}|)$. It also requires storing the intermediate estimated truth degrees for all L steps for all N rules to calculate the gradients, resulting in a space complexity of $\mathcal{O}(NL|\mathcal{E}|)$. Besides, the space complexity of the predicate selection weight is $\mathcal{O}(NL|\mathcal{R}|)$. Therefore, the space complexity of a forward computation step for TensorLog is $\mathcal{O}(NL(|\mathcal{R}|+|\mathcal{E}|+|\mathcal{K}|))$.

(II) We then prove that the space complexity of a backward propagation step for FastLog is $\mathcal{O}(NL(|\mathcal{R}|+|\mathcal{E}|+|\mathcal{K}|))$. Let $z = \mathcal{F}_{\mathrm{e2f}}(\phi_{r,x}^{(k,l-1)}) \odot \mathcal{F}_{\mathrm{r2f}}(w^{(r,k,l)})$. The space complexity for calculating $\frac{\partial \mathcal{F}_{\mathrm{f2e}}(z)}{\partial z}$ is $\mathcal{O}(|\mathcal{E}|+|\mathcal{K}|)$. The space complexity for calculating $\frac{\partial z}{\partial \mathcal{F}_{\mathrm{e2f}}(\phi_{r,x}^{(k,l-1)})}$ is $\mathcal{O}(|\mathcal{K}|)$. The space complexity for calculating $\frac{\partial z}{\partial \mathcal{F}_{\mathrm{r2f}}(w^{(r,k,l)})}$ is $\mathcal{O}(|\mathcal{K}|)$. The space complexity for calculating $\frac{\partial \mathcal{F}_{\mathrm{r2f}}(w^{(r,k,l)})}{\partial w^{(r,k,l)}}$ is $\mathcal{O}(|\mathcal{K}|+|\mathcal{R}|)$. Therefore, the space complexity of a backward propagation step for FastLog is $\mathcal{O}(NL(|\mathcal{R}| + |\mathcal{E}| + |\mathcal{K}|))$. □

E. Proof of Proposition 5

Proposition 5. *For an arbitrary triple* $(a, r, b) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, $\forall N \geq 1, L \geq 1$: $\texttt{TensorLog}(\theta_r^{N,L}, a, b) = \texttt{FastLog}(\theta_r^{N,L}, a, b)$.

Proof. To prove Proposition 5, 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

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

□

F. Proof of Proposition 6

Proposition 6. *The time complexity for the training phase of* $\texttt{FastLog}^{c_1,c_2}$ *is* $\mathcal{O}(NLc_2)$.

Proof. (I) We first prove that the time complexity of a forward computation step for $\texttt{FastLog}^{c_1,c_2}$ is $\mathcal{O}(NLc_2)$. From Proposition 2, we know that the time complexity of a forward computation step for FastLog is $\mathcal{O}(NL|\mathcal{K}|)$. From Equations (10-11), we know that only top-c_2 intermediate hidden states involve floating-point multiplications and additions. Therefore, the time complexity of a forward computation step for $\texttt{FastLog}^{c_1,c_2}$ is $\mathcal{O}(NLc_2)$.

(II) We then prove that the time complexity of a backward propagation step for FastLog is $\mathcal{O}(NLc_2)$. From Equations (9), we know that only top-c_1 intermediate estimated truth degrees are used to calculate the gradients. From Equations (10), we know that only top-c_2 intermediate hidden states are used to calculate the gradients. Let $z = \hat{\mathcal{F}}_{\text{r2f}}^{c_2}(\hat{\mathcal{F}}_{\text{e2f}}^{c_1}(\phi_{r,x}^{(k,l-1)}), w^{(r,k,l)})$. 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 $\hat{\mathcal{F}}_{\text{e2f}}^{c_1}(\phi_{r,x}^{(k,l-1)})$ are used to calculate gradients. The time complexity for calculating $\frac{\partial \hat{\mathcal{F}}_{\text{e2f}}^{c_1}(\phi_{r,x}^{(k,l-1)})}{\partial \phi_{r,x}^{(k,l-1)}}$ is $\mathcal{O}(c_2)$ because only the top-c_2 elements in $\hat{\mathcal{F}}_{\text{e2f}}^{c_1}(\phi_{r,x}^{(k,l-1)})$ are used to calculate gradients. Therefore, the time complexity of a backward propagation step for FastLog is $\mathcal{O}(NLc_2)$. □

G. Proof of Proposition 7

Proposition 7. *The space complexity for the training phase of* $\texttt{FastLog}^{c_1,c_2}$ *is* $\mathcal{O}(NL(c_1 + c_2 + |\mathcal{R}|))$.

Proof. (I) We first prove that the space complexity of a forward computation step for $\texttt{FastLog}^{c_1,c_2}$ is $\mathcal{O}(NL(c_1 + c_2 + |\mathcal{R}|))$. For a forward computation step, $\texttt{FastLog}^{c_1,c_2}$ 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. Besides, the space complexity of the predicate selection weight is $\mathcal{O}(NL|\mathcal{R}|)$. Therefore, the space complexity of a forward computation step for $\texttt{FastLog}^{c_1,c_2}$ is $\mathcal{O}(NL(c_1 + c_2 + |\mathcal{R}|))$.

(II) We then prove that the space complexity of a backward propagation step for $\texttt{FastLog}^{c_1,c_2}$ is $\mathcal{O}(NL(c_1 + c_2 + |\mathcal{R}|))$. Let $z = \hat{\mathcal{F}}_{\text{r2f}}^{c_2}(\hat{\mathcal{F}}_{\text{e2f}}^{c_1}(\hat{\phi}_{r,x}^{(k,l-1)}), w^{(r,k,l)})$. The space complexity for calculating $\frac{\partial \hat{\mathcal{F}}_{\text{f2e}}(z)}{\partial z}$ is $\mathcal{O}(c_1 + c_2)$. The space complexity for calculating $\frac{\partial z}{\partial \hat{\mathcal{F}}_{\text{e2f}}(\hat{\phi}_{r,x}^{(k,l-1)})}$ is $\mathcal{O}(c_2)$. The space complexity for calculating $\frac{\partial z}{\partial w^{(r,k,l)}}$ is $\mathcal{O}(c_2 + |\mathcal{R}|)$. Therefore, the space complexity of a backward propagation step for $\texttt{FastLog}^{c_1,c_2}$ is $\mathcal{O}(NL(c_1 + c_2 + |\mathcal{R}|))$. □

H. Proof of Proposition 8

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

Proof. From Equations (4-6) and (9-11), we know that $\hat{\mathcal{F}}_{\text{e2f}}^{|\mathcal{E}|}$ (resp. $\hat{\mathcal{F}}_{\text{r2f}}^{|\mathcal{K}|}$ 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 (7) can be derived by:

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

□

II. 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`.

A. The *NeuralLP* Method

`NeuralLP` [1] 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 [2] 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

$$
\begin{aligned}
h_l &= \text{LSTM}(h_{l-1}, q_l), \\
w^{(r,1,l)} &= \text{Softmax}(W h_l + b),
\end{aligned}
\tag{14}
$$

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

$$
\alpha^{(r,1,l)} = \text{Softmax}([h_0^\top h_l; h_1^\top h_l; \cdots; h_{l-1}^\top h_l]), \tag{15}
$$

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

$$
\phi_{r,x}^{(1,l)} = \begin{cases} \sum_{i=1}^{2n} (\sum_{j=0}^{l-1} \alpha_j^{(r,1,l)} \phi_{r,x}^{(1,j)})(w_i^{(r,1,l)} M_{r_i}), & 1 \leq l \leq L, \\ \sum_{j=0}^{L} \alpha_j^{(r,1,L+1)} \phi_{r,x}^{(1,j)}, & l = L+1, \end{cases} \tag{16}
$$

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

$$
\texttt{NeuralLP}(\theta_r^{1,L}, x, y) = \phi_{r,x}^{(1,L+1)} v_y, \tag{17}
$$

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 for the training phase of `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 (14-17), 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 (14-17), 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 $\mathcal{O}(L(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|) + L^2|\mathcal{E}|)$. □

B. The *DRUM* Method

Different from `NeuralLP`, `DRUM` [3] 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

$$\begin{aligned} \overrightarrow{h}_{l}^{(k)} &= \overrightarrow{\text{BiLSTM}}^{(k)}(\overrightarrow{h}_{l-1}^{(k)}, v_r), \\ \overleftarrow{h}_{L-l+1}^{(k)} &= \overleftarrow{\text{BiLSTM}}^{(k)}(\overleftarrow{h}_{L-l}^{(k)}, v_r), \\ w^{(r,k,l)} &= \text{Softmax}(W[\overrightarrow{h}_{l}^{(k)}; \overleftarrow{h}_{L-l+1}^{(k)}] + b), \end{aligned} \tag{18}$$

where both $\overrightarrow{h}_{0}^{(k)}$ and $\overleftarrow{h}_{L+1}^{(k)}$ 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 $\phi_{r,x}^{(k,l)} \in \mathbb{R}^{|\mathcal{E}|}$ for `DRUM` are estimated by

$$\phi_{r,x}^{(k,l)} = \sum_{i=1}^{2n+1} \phi_{r,x}^{(k,l-1)} (w_i^{(r,k,l)} M_{r_i}), \tag{19}$$

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

$$\texttt{DRUM}(\theta_r^{N,L}, x, y) = \sum_{k=1}^{N} \phi_{r,x}^{(k,L)} v_y, \tag{20}$$

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

C. The *smDRUM* Method

`smDRUM` [4] 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 $\phi_{r,x}^{(k,l)} \in \mathbb{R}^{|\mathcal{E}|}$ for `smDRUM` are estimated by

$$\phi_{r,x}^{(k,l)} = \sum_{i=1}^{2n+1} \phi_{r,x}^{(k,l-1)} \otimes (w_i^{(r,k,l)} M_{r_i}), \tag{21}$$

where $\phi_{r,x}^{(k,0)} = v_x^\top$, $\otimes$ 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 $1 \leq i \leq a$ and $1 \leq j \leq b$. The truth degree of (x,r,y) is estimated by

$$\texttt{smDRUM}(\theta_r^{N,L}, x, y) = \sum_{k=1}^{N} \phi_{r,x}^{(k,L)} v_y, \tag{22}$$

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

D. The *mmDRUM* Method

`mmDRUM` [4] 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 $\phi_{r,x}^{(k,l)} \in \mathbb{R}^{|\mathcal{E}|}$ for `mmDRUM` are estimated by

$$\phi_{r,x}^{(k,l)} = \sum_{i=1}^{2n+1} \phi_{r,x}^{(k,l-1)} \otimes (w_i^{(r,k,l)} M_{r_i}), \tag{23}$$

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

$$\texttt{mmDRUM}(\theta_r^{N,L}, x, y) = \max_{k=1}^{N} \phi_{r,x}^{(k,L)} v_y, \tag{24}$$

where $\theta_r^{N,L} = \{v_r, W, b\} \cup \bigcup_{1\leq k\leq N} \theta_{\text{BiLSTM}}^{(k)}$ 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 for the training phase of `DRUM` , `smDRUM`, and `mmDRUM` is $\mathcal{O}(NL(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|))$.*

Proof. From Equations (19-24), 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 (18-20), we know that the time complexity of a forward computation step for `DRUM` is

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

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 (18-20), 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 $\mathcal{O}(NL(|\mathcal{K}| + |\mathcal{R}||\mathcal{E}|))$. □

III. 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`.

A. 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 (14-15) 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

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

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

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

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

Proposition 11. *The time complexity of for the training phase of 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 FastLog is $\mathcal{O}(NL|\mathcal{K}|)$. From Equations (25-26), we know that the time complexity of a forward computation step for NeuralLP-FL is

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

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 $(a, r, b) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, $\forall L \geq 1$: NeuralLP$-$FL$(\theta_r^{1,L}, a, b)$ = 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 $1 \leq l \leq L$, 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

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

□

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

B. 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 (18) for selection weight estimation. 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

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

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 for the training phase of* 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* $(a,r,b) \in \mathcal{E}\times\mathcal{R}\times\mathcal{E}$, $\forall N \geq 1, L \geq 1$: *DRUM*$-\text{FL}(\theta_r^{N,L}, a, b) =$ *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 $M_{\text{e2f}} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{K}|}$ (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

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

□

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

C. The *smDRUM*-FL Method

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

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

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

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

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

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

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* $(a,r,b) \in \mathcal{E}\times\mathcal{R}\times\mathcal{E}$, $\forall N \geq 1, L \geq 1$: *smDRUM*$-\text{FL}(\theta_r^{N,L}, a, b) =$ *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 $M_{\text{e2f}} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{K}|}$ (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

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

Therefore, we have

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

□

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

D. The *mmDRUM*-FL Method

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

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

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

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

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* $(a, r, b) \in \mathcal{E} \times \mathcal{R} \times \mathcal{E}$, $\forall N \geq 1, L \geq 1$: *mmDRUM*$\text{−FL}(\theta_r^{N,L}, a, b) =$ *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 $M_{\text{e2f}} \in \mathbb{R}^{|\mathcal{E}|\times|\mathcal{K}|}$ (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

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

Therefore, we have

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

□

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

IV. Detailed Experimental Setting

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 [5]. Following the work [6], 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. It is worth noting that in the preliminary setting for ranking, the rank of a correct answer is set to $j+1$. As pointed out by [6], [7], this setting for ranking is problematic because in some algorithms the correct answer can be assigned the same truth degrees as other entities. For example, if an algorithm predicts all truth degrees to 0, then the rank for any answer will be computed as 1, leading to a wrong assessment of the algorithm. Therefore, we used $j + (k+1)/2$ to calculate the ranks for a more reasonable assessment.

Note that there are some existing methods such as AnyBURL [8] calculate the rank of a correct answer by sorting all corrupted triples based on their estimated truth degrees. This way will not produce the same ranking for triples that have the same truth degrees, but the ranking relies on the serial numbers of entities. To mitigate the dependence of the serial numbers of entities, we also used $j + (k+1)/2$ to calculate the ranks for AnyBURL.

Implementation Details. We implemented FastLog[1] 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

[1]Code and data, along with supplementary materials, are available at: https://github.com/qikunxun/FastLog.

TABLE I: Hyper-parameter settings for NeuralLP, DRUM, smDRUM and mmDRUM on different datasets.

Method	Hyper-parameter	Dataset							
		Family	Kinship	UMLS	WN18RR	FB15k-237	YAGO3-10	Wikidata5M	Freebase
NeuralLP-FL	Maximum number of rules N	1	1	1	1	1	1	1	1
	Maximum size of body atoms L	3	3	3	3	3	3	3	3
	Batch size	64	64	64	64	64	64	32	2
	Learning rate	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3
	Hidden size d for estimating weights	128	128	128	128	128	128	128	128
	Optimizer	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam
DRUM-FL, smDRUM-FL, mmDRUM-FL	Maximum number of rules N	3	3	3	3	3	3	3	3
	Maximum size of body atoms L	3	3	3	3	3	3	3	3
	Batch size	64	64	64	64	64	64	32	2
	Learning rate	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3	1e-3
	Hidden size d for estimating weights	128	128	128	128	128	128	128	128
	Optimizer	Adam	Adam	Adam	Adam	Adam	Adam	Adam	Adam

TABLE II: Hyper-parameter settings for NeuralLP, DRUM, smDRUM and mmDRUM on different datasets.

Method	Hyper-parameter	Dataset	
		FB15k-237 (Inductive)	NELL-995 (Inductive)
NeuralLP-FL	Maximum number of rules N	1	1
	Maximum size of body atoms L	3	3
	Batch size	64	64
	Learning rate	1e-3	1e-3
	Hidden size d for estimating weights	128	128
	Optimizer	Adam	Adam
DRUM-FL, smDRUM-FL, mmDRUM-FL	Maximum number of rules N	3	3
	Maximum size of body atoms L	3	3
	Batch size	64	64
	Learning rate	1e-3	1e-3
	Hidden size d for estimating weights	128	128
	Optimizer	Adam	Adam

TABLE III: Hyper-parameter settings for DRUM-FL+ on different datasets.

Hyper-parameter	Dataset			
	UMLS	FB15k-237	Wikidata5M	Freebase
Maximum number of rules N	100	3	10	3
Maximum size of body atoms L	3	3	3	3
Training time	7.3m	1.5h	24h	22.2h
Batch size	64	64	16	2
Learning rate	1e-3	1e-3	3e-4	1e-3
Hidden size d for estimating predicate selection weights	128	128	2048	128

24GB memory. Note that we require 1TB RAM to reproduce the results of AnyBURL, as AnyBURL requires 900GB RAM to learn rules from Freebase [8]. FastLog only requires a maximum of 25GB RAM for training and evaluation.

To ensure a fair comparison, we used the original hyper-parameter settings for baseline methods on Family, Kinship, UMLS, WN18RR, FB15k-237, YAGO3-10, and the two datasets under the inductive setting. In more detail, all baseline methods and the FastLog-enhanced methods were trained by Adam [9] with 10 training epochs. An early stopping strategy was applied to maximize the Hit@10 score on the validation set. The initial learning rate was set to 1e-3, and the mini-batch size was set to 64. The maximum length of rules L was set to 3, and the maximum number of rules to be learnt N was set to 3 for DRUM, smDRUM, mmDRUM. For the large-scale datasets Wikidata5M and Freebase, we employed a sampling strategy to ensure efficient training. Specifically, we sampled a mini-batch of triples for each step in the training phase, where the mini-batch size was set to 32 for Wikidata5M and 2 for Freebase. The training phase was stopped until it triggered the stopping condition, e.g., exceeding the user-specified training time limit. For all datasets and all methods, the hidden size for estimating the predicate selection weight were set as 128. Detailed settings for all hyper-parameters are shown in Table I and Table II.

To help reproduce our results for DRUM-FL+, we provide the detailed hyper-parameter settings used in our experiments. Table III reports the detailed hyper-parameter settings for DRUM-FL+ in regard to different datasets.

Baselines. We compare the FastLog-enhanced methods with the following search-based rule learning methods:

- AMIE [10] is a well-known search-based method that

TABLE IV: Analysis on inference consistency across all baselines and all metrics.

Method	Train	Test	Family				Kinship				UMLS			
			MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10	MRR	H@1	H@3	H@10
NeuralLP	TensorLog	TensorLog	0.923	87.1	97.2	98.7	0.468	30.4	54.7	82.6	0.686	53.3	81.0	93.0
	TensorLog	FastLog	0.923	87.1	97.2	98.7	0.468	30.4	54.7	82.6	0.686	53.3	81.0	93.0
	FastLog	FastLog	0.926	87.5	97.4	98.8	0.472	30.5	55.3	84.3	0.707	55.1	84.1	93.6
	FastLog	TensorLog	0.926	87.5	97.4	98.8	0.472	30.5	55.3	84.3	0.707	55.1	84.1	93.6
DRUM	TensorLog	TensorLog	0.941	89.8	98.2	99.0	0.471	30.0	55.0	84.5	0.706	56.1	82.1	93.9
	TensorLog	FastLog	0.941	89.8	98.2	99.0	0.471	30.0	55.0	84.5	0.706	56.1	82.1	93.9
	FastLog	FastLog	0.951	92.0	98.0	99.0	0.475	30.4	55.5	85.5	0.742	60.3	86.3	94.7
	FastLog	TensorLog	0.951	92.0	98.0	99.0	0.475	30.4	55.5	85.5	0.742	60.3	86.3	94.7
smDRUM	TensorLog	TensorLog	0.957	92.6	98.4	99.0	0.425	25.1	49.8	82.1	0.738	60.1	84.8	94.3
	TensorLog	FastLog	0.957	92.6	98.4	99.0	0.425	25.1	49.8	82.1	0.738	60.1	84.8	94.3
	FastLog	FastLog	0.959	93.0	98.4	99.0	0.439	26.3	51.5	84.2	0.744	61.4	85.0	94.4
	FastLog	TensorLog	0.959	93.0	98.4	99.0	0.439	26.3	51.5	84.2	0.744	61.4	85.0	94.4
mmDRUM	TensorLog	TensorLog	0.904	83.0	96.9	98.9	0.286	13.0	30.7	66.8	0.465	31.8	52.0	79.3
	TensorLog	FastLog	0.904	83.0	96.9	98.9	0.286	13.0	30.7	66.8	0.465	31.8	52.0	79.3
	FastLog	FastLog	0.926	86.0	97.8	99.0	0.304	13.3	31.0	68.4	0.478	32.9	54.1	78.3
	FastLog	TensorLog	0.926	86.0	97.8	99.0	0.304	13.3	31.0	68.4	0.478	32.9	54.1	78.3

introduces the partial completeness assumption (PCA) to learn logical rules. We compare the latest version of AMIE, namely AMIE3 [11], in our evaluation.

- AnyBURL [8] is a SOTA search-based method. It learns both chain-like rules and logical rules with entity constants to improve efficacy.

We also compare the FastLog-enhanced methods with the following neural-based rule learning methods:

- RNNLogic [6] is a SOTA neural-based rule learner that exploits a neural rule generator to yield high-quality rules based on the mined relational paths and the test scores.
- RLogic [12] is a SOTA approach that exploits a predicate representation scoring model to induce chain-likes rules with high evaluation scores.
- NCRL [13] is a SOTA rule learner that improves the learning efficiency by decomposing the induction of chain-like rules into the learning of shorter sub-rules.

For a more comprehensive comparison, we compare the FastLog-enhanced methods with the following embedding-based and PLM-based methods:

- Complex [14] is an embedding-based method that represents entities and relations as complex-valued vectors.
- RotatE [15] is a SOTA embedding-based approach that models relations as rotations in the complex plane.
- KGT5 [16] is a SOTA PLM-based approach that leverages the pre-trained language model T5 [17] to model the entities and relations in a KG.

Furthermore, we also compare the following GNN-based methods.

- NBFNet [18] models link prediction by parameterizing the Bellman-Ford algorithm with three functions: "INDICATOR" for boundary initialization, "MESSAGE" for propagating path information, and "AGGREGATE" for combining paths. This path-based design unifies classical heuristics and achieves strong efficacy in both transductive and inductive settings.
- RED-GNN [19] leverages relational directed graphs (r-digraphs) to capture complex, overlapping relational paths for efficient KG reasoning. By utilizing dynamic programming to recursively encode multiple r-digraphs, it achieves SOTA efficacy in both inductive and transductive settings while maintaining high interpretability.
- A*Net [20] is a scalable path-based method for KG reasoning that uses a learned priority function to selectively visit important nodes and edges, improving efficiency while maintaining competitive efficacy in link prediction.
- AdaProp [21] learns adaptive propagation paths for KG reasoning, addressing the issues of irrelevant entities and exponential growth in propagation steps.

V. Additional Experiments

Our Proposition 5 establishes the equivalence between FastLog and TensorLog inference. However, Table IV and Table V in the main text report slight differences in efficacy scores. These variations arise from the randomness involved in training, such as the different orderings of data sampling and parameter initialization. This issue remains even when the same random seeds have been used, since we rely on the original baseline implementations. To further validate Proposition 5, we froze the parameters of DRUM trained by TensorLog (resp. FastLog) and performed inference with FastLog (resp. TensorLog). As shown in Table IV, all evaluation metrics across four baseline methods and three datasets became identical under this setting. These results affirm the correctness of Proposition 5, implying that FastLog is able to improve the efficiency of existing end-to-end rule learning methods without efficacy degradation.

TABLE V: Examples of rules learnt by different systems.

Method	Logical rules and reasoning paths	New Fact	Label
DRUM-FL	R_1 : nephewOf(x, y) ← brotherOf(x, z) ∧ nephewOf(z, y) P_1 : nephewOf(ENT95, ENT40) ⇐ brotherOf(ENT95, ENT151)∧ nephewOf(ENT151, ENT40)	(ENT95, nephewOf, ENT40)	True
	R_2 : husbandOf(x, y) ← sonOf$^-(x, z)$ ∧ sonOf$^-(z, y)$ P_2 : husbandOf(ENT599, ENT600) ⇐ sonOf$^-$(ENT599, ENT1124)∧ sonOf(ENT1124, ENT600)	(ENT599, husbandOf, ENT600)	False
AnyBURL	R_3 : nieceOf(x, y) ← sisterOf(x, z) ∧ nieceOf(z, y) P_3 : nieceOf(ENT99, ENT11) ⇐ sisterOf(ENT99, ENT5) ∧ nieceOf(ENT5, ENT11)	(ENT99, nieceOf, ENT11)	True
	R_4 : auntOf(x, y) ← nieceOf$^-(x, y)$ P_4 : auntOf(ENT993, ENT999) ⇐ nieceOf$^-$(ENT993, ENT999)	(ENT993, husbandOf, ENT999)	False
	R_5 : auntOf$(x,$ ENT99$)$ ← nieceOf$^-(x,$ ENT5$)$ P_5 : auntOf(ENT11, ENT99) ⇐ nieceOf$^-$(ENT11, ENT5)	(ENT11, auntOf, ENT99)	True

VI. Case Study

A. Error Analysis on Learnt Rules

To further clarify the limitation of FastLog, we conducted an error analysis on the Family dataset. Specifically, we applied the rule extraction algorithm proposed in the work [4] to extract the learnt rules of DRUM-FL. Furthermore, we also compare the rules learnt by AnyBURL. Table V illustrates five examples of learnt rules, along with a reasoning path for each rule. We can observe that both DRUM-FL and AnyBURL are able to learn precise rules (e.g., R_1 and R_3) for reasoning. However, both of them still learn imprecise rules such as R_2 and R_4. In more detail, both R_2 and R_4 produce incorrect new facts because these two rules lack type constraints on entity x. For example, R_2 (resp. R_4) can avoid inferring incorrect new facts by adding a type constraint Male(x) (resp. Female(x)) to the rule body. These cases highlight a limitation of both AnyBURL and DRUM-FL. That is, the chain-like rules learnt by them are insufficiently specialized. Therefore, our future work will focus on learning more specialized and expressive rules based on the efficient FastLog framework.

Besides, rule R_5 illustrates a logical rule with entity constants learned by AnyBURL. This rule demonstrates how AnyBURL can achieve higher efficacy in certain reasoning scenarios by capturing specialized patterns from the training data. While such rules may improve accuracy, their applicability remains limited. Specifically, incorporating entity constants in the rule body tends to overfit the training data, thereby reducing effectiveness in inductive reasoning tasks. This example highlights why AnyBURL fails to generalize well to the inductive setting in our evaluation.

B. Case Study on LLM Integration

As mentioned in our motivations, learning high-quality logical rules can help improve the reasoning ability of LLMs. To verify this, we conducted a case study on enhancing the well-known LLM GPT-4o by the rules learnt by the FastLog-enhanced methods. In more detail, we investigate whether GPT-4o can answer test queries in the UMLS dataset, which is a KG in the field of biomedical. Figure 1 (a) shows that GPT-4o can respond to a sampled query but the answers are unseen entities in UMLS. Figure 1 (b) shows that we can enhance the domain knowledge of GPT-4o by promoting it with retrieved facts from the training set, where the retrieved facts are selected from the two-hop subgraphs on the entity "eicosanoid" in UMLS. However, we observed that GPT-4o merely reproduces the provided facts without performing reasoning beyond them. In contrast, as shown in Figure 1(c), equipping GPT-4o with a logical rule learned by DRUM-FL enables it to infer missing facts from the retrieved facts by understanding and applying the rule extracted from DRUM-FL, thereby yielding more accurate and informative answers. These cases highlight the application value of FastLog.

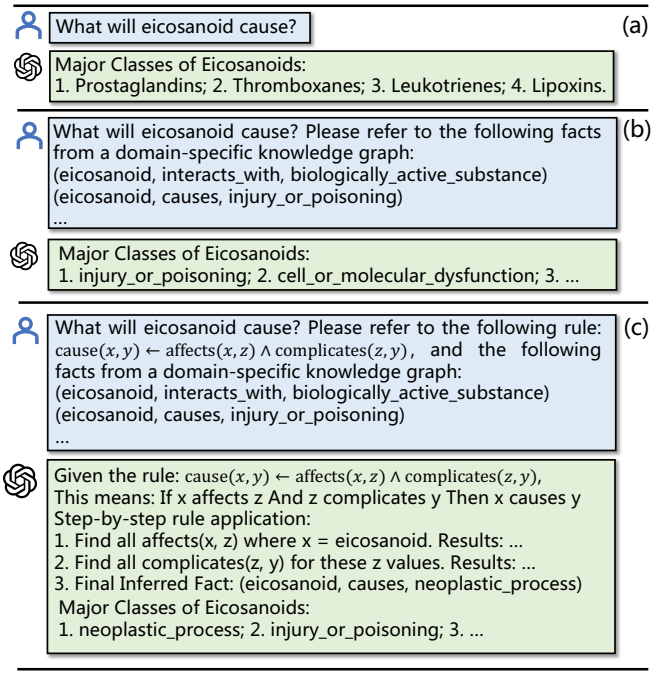

Fig. 1: Case study on enhancing LLMs with learnt rules.

References

[1] F. Yang, Z. Yang, and W. W. Cohen, "Differentiable learning of logical rules for knowledge base reasoning," in *NIPS*, 2017, pp. 2319–2328.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] A. Sadeghian, M. Armandpour, P. Ding, and D. Z. Wang, "DRUM: end-to-end differentiable rule mining on knowledge graphs," in *NeurIPS*, 2019, pp. 15 321–15 331.

[4] X. Wang, D. J. T. Cucala, B. C. Grau, and I. Horrocks, "Faithful rule extraction for differentiable rule learning models," in *ICLR*, 2024.

[5] A. Bordes, N. Usunier, A. García-Durán, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *NIPS*, 2013, pp. 2787–2795.

[6] M. Qu, J. Chen, L. A. C. Xhonneux, Y. Bengio, and J. Tang, "Rnnlogic: Learning logic rules for reasoning on knowledge graphs," in *ICLR*, 2021.

[7] Z. Sun, S. Vashishth, S. Sanyal, P. P. Talukdar, and Y. Yang, "A re-evaluation of knowledge graph completion methods," in *ACL*, 2020, pp. 5516–5522.

[8] C. Meilicke, M. W. Chekol, P. Betz, M. Fink, and H. Stuckenschmidt, "Anytime bottom-up rule learning for large-scale knowledge graph completion," *VLDB J.*, vol. 33, no. 1, pp. 131–161, 2024.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.

[10] L. A. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek, "AMIE: association rule mining under incomplete evidence in ontological knowledge bases," in *WWW*, 2013, pp. 413–422.

[11] J. Lajus, L. Galárraga, and F. M. Suchanek, "Fast and exact rule mining with AMIE 3," in *ESWC*, vol. 12123, 2020, pp. 36–52.

[12] K. Cheng, J. Liu, W. Wang, and Y. Sun, "Rlogic: Recursive logical rule learning from knowledge graphs," in *KDD*. ACM, 2022, pp. 179–189.

[13] K. Cheng, N. K. Ahmed, and Y. Sun, "Neural compositional rule learning for knowledge graph reasoning," in *ICLR*, 2023.

[14] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard, "Complex embeddings for simple link prediction," in *ICML*, vol. 48, 2016, pp. 2071–2080.

[15] Z. Sun, Z. Deng, J. Nie, and J. Tang, "Rotate: Knowledge graph embedding by relational rotation in complex space," in *ICLR*, 2019.

[16] A. Saxena, A. Kochsiek, and R. Gemulla, "Sequence-to-sequence knowledge graph completion and question answering," in *ACL*, 2022, pp. 2814–2828.

[17] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res. (JMLR)*, vol. 21, pp. 140:1–140:67, 2020.

[18] Z. Zhu, Z. Zhang, L. A. C. Xhonneux, and J. Tang, "Neural bellman-ford networks: A general graph neural network framework for link prediction," in *NeurIPS*, 2021, pp. 29 476–29 490.

[19] Y. Zhang and Q. Yao, "Knowledge graph reasoning with relational digraph," in *WWW*, 2022, pp. 912–924.

[20] Z. Zhu, X. Yuan, M. Galkin, L. A. C. Xhonneux, M. Zhang, M. Gazeau, and J. Tang, "A*net: A scalable path-based reasoning approach for knowledge graphs," in *NeurIPS*, 2023.

[21] Y. Zhang, Z. Zhou, Q. Yao, X. Chu, and B. Han, "Adaprop: Learning adaptive propagation for graph neural network based knowledge graph reasoning," in *KDD*, 2023, pp. 3446–3457.