Improving Neural Logic Machines via Failure Reflection

Zhiming Li [*1] **Yushi Cao** [*1] **Yan Zheng** [2] **Xu Liu** [3] **Bozhi Wu** [1] **Tianlin Li** [1] **Xiufeng Xu** [1] **Junzhe Jiang** [4]
Yon Shin Teo [5] **Shang-wei Lin** [1] **Yang Liu** [1]

Abstract

Reasoning is a fundamental ability towards artificial general intelligence (AGI). Fueled by the success of deep learning, the neural logic machines models (NLMs) have introduced novel neural-symbolic structures and demonstrate great performance and generalization on reasoning and decision-making tasks. However, the original training approaches of the NLMs are still far from perfect, the models would repeat similar mistakes during the training process which leads to sub-optimal performance. To mitigate this issue, we present a novel framework named Failure Reflection Guided Regularizer (FRGR). FRGR first dynamically identifies and summarizes the root cause if the model repeats similar mistakes during training. Then it penalizes the model if it makes similar mistakes in future training iterations. In this way, the model is expected to avoid repeating errors of similar root causes and converge faster to a better-performed optimum. Experimental results on multiple relational reasoning and decision-making tasks demonstrate the effectiveness of FRGR in improving performance, generalization, training efficiency, and data efficiency. Our code is available at `https://sites.google.com/view/frgr-icml24`.

1. Introduction

Neural-symbolic AI (Garcez et al., 2022; Susskind et al., 2021; Evans & Grefenstette, 2018) has become a novel and active research direction towards better reasoning ability, which is critical for artificial general intelligence (Lake et al., 2017; Kojima et al., 2022). To achieve this, researchers seek to combine the strengths of both deep learning and symbolic approaches (logic reasoning). Following this trend, neural logic machine models (NLMs) (Zimmer et al., 2023; Dong et al., 2018) have been proposed to learn logic programs via gradient-based optimization. These NLMs introduce inductive bias to build neural-symbolic structures that realize Horn clauses (Horn, 1951) in first-order logic. It is surprising that even though these NLMs are small in size, thanks to their logical design, they have shown great performance and generalization on the reasoning tasks (e.g., decision making, relational reasoning) even powerful Large language models (LLMs) fail (Valmeekam et al., 2023; Li et al., 2024). Besides, compared to SAT solver-based methods (Muggleton et al., 2015; Raghothaman et al., 2019), the NLMs do not require well-crafted human expert bias for search space construction (i.e., carefully designed metarules or templates (Muggleton et al., 2015)), which makes them easier to apply.

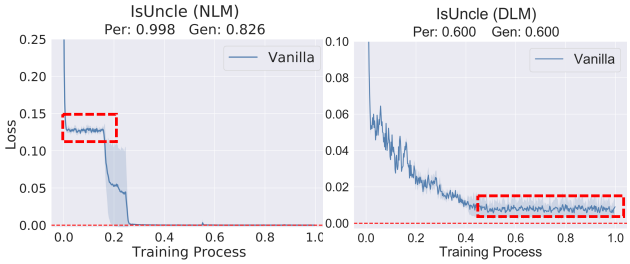

Figure 1. Training loss curve, performance (Per), and generalization (Gen) of the NLM, DLM models optimized with original training approach on the `IsUncle` reasoning task.

However, despite the accomplishment of the NLMs, we observe that the original training approach of the current state-of-the-art models (Dong et al., 2018; Zimmer et al., 2023) are ineffective and often make the models converge to an ill-performed local optimum with high oscillation. Figure 1 shows the training loss curves of two state-of-the-art neural logic machine models (neural logic machines (NLM) (Dong et al., 2018) and differentiable logic machines (DLM) (Zimmer et al., 2023)) on the `IsUncle` reasoning

[*]Equal contribution [1]Nanyang Technological University, Singapore [2]Tianjin university, Tianjin, China [3]National University of Singapore, Singapore [4]Hong Kong Polytechnic University, Hong Kong [5]Continental Automotive Singapore Pte. Ltd., Singapore. Correspondence to: Yan Zheng <yanzheng@tju.edu.cn>, Tianlin Li <tianlin001@e.ntu.edu.sg>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

task[1]. This task requires the model to learn the logic that deduces whether a person y is another person x's uncle in a family tree based on basic input father/mother, and son/daughter relations. We can observe that the NLM model oscillates for a long while before the loss drops again (highlighted with the red bounding box), and the DLM model keeps oscillating around a local optimum and fails to achieve optimal results on the training data. A natural explanation for this phenomenon is that the model is repeating mistakes with similar root cause subprograms and the original training approach fails to identify it and stop the model from doing so (see Section 4 for the motivation validation).

Based on the above intuition, we present a novel framework called Failure Reflection Guided Regularizer (FRGR). The key idea of FRGR is to dynamically identify and summarize the root cause once the model starts repeating mistakes of a similar kind. Then based on the summarized root cause, FRGR penalizes the model if it repeats similar mistakes in future training iterations. With FRGR, the model is expected to jump out of the oscillation faster and converge to a better-performed optimum.

To demonstrate the effectiveness of FRGR, we apply FRGR on the previous state-of-the-art NLMs on the inductive logic programming (ILP) tasks (Zimmer et al., 2023; Dong et al., 2018; Evans & Grefenstette, 2018). Specifically, we conduct experiments on both the NLM and DLM models under the ideal data-rich setting whose demonstrating examples are abundant. We also evaluate our approach under a simulated data-scarce setting to see whether it is effective for low-resource scenarios whose demonstrating examples are limited. Besides, to understand whether FRGR has mitigated models' erroneous behavior, we conduct an in-depth analysis of the models' behavior during the training process. We show that FRGR is effective in improving the performance, generalization, and training efficiency under both the data-rich and data-scarce settings by effectively mitigating models' erroneous behavior. The contributions of this work are three-fold:

- We propose a novel regularization approach called FRGR to optimize the learning process of the NLMs. The idea is to timely detect models' repetition of similar errors, identify the root cause subprogram, and use it to regularize the model.
- Experimental results on two current state-of-the-art NLMs demonstrate that FRGR can significantly improve the models' performance, generalization, and training efficiency under both the data-rich and data-scarce scenarios.
- To the best of our knowledge, it is the first paper to demonstrate the effectiveness of utilizing error root causes to improve the neural network models, which may beneficially motivate the community to extend the idea to other domains beyond ILP tasks.

[1]Note that this is a challenging reasoning task even the state-of-the-art GPT4-Turbo model performs poorly (Li et al., 2024).

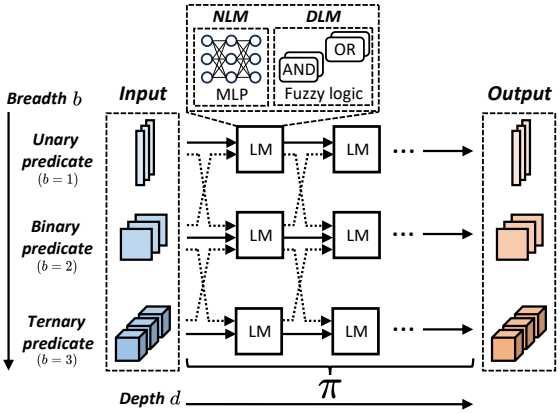

Figure 2. Model architecture of the NLMs.

2. Preliminary

In this section, we introduce the fundamentals of inductive logic programming and neural logic machine models.

2.1. Inductive Logic Programming

Logic programming (LP) is a programming paradigm based on first-order logic (FOL). There are two basic primitives for FOL: *predicate* and *variable*. A predicate p denotes the name of a property/relation verification function $p(v_1, \ldots, v_n)$, also called *atom*, where $v_1, \ldots, v_n$ are the input variables of this function. *E.g.*, for atom $\texttt{IsMother}(x, y)$, $\texttt{IsMother}$ is the predicate, and the atom takes two variables x, y as input. An atom is called a *ground atom* if all the variables of it are instantiated with constants. For example, we instantiate x, y of $\texttt{IsMother}(x, y)$ with two people A, B respectively, if B is A's mother, then $\texttt{IsMother}(A, B) = \texttt{True}$, otherwise the result would be $\texttt{False}$. Then, based on the atoms, we can construct the FOL rule, which is in the form: $\alpha \leftarrow \alpha_1 \odot \cdots \odot \alpha_n$, $\odot$ denotes logical operators (i.e., conjunction, disjunction). Inductive Logic Programming (ILP) (Getoor & Taskar, 2007; Muggleton, 1991; Getoor et al., 2001) refers to the problem of learning a logic program (written in FOL) given a set of demonstrating examples. The learned program is required to deduce all the positive examples and none of the negative examples.

2.2. NLMs

The neural logic machines (NLM) (Dong et al., 2018) and differentiable logic machines (DLM) (Zimmer et al., 2023)

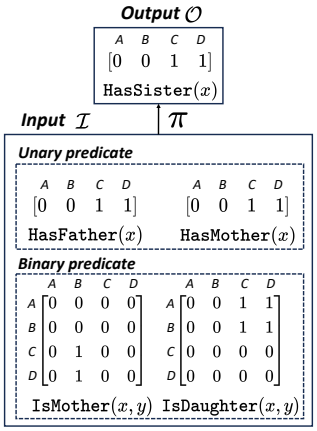

Figure 3. Truth value tensor of the `HasSister` task sample. Given the truth value tensor of the input predicates `HasMother`, `HasFather`, `IsMother` and `IsDaughter`, the neural logic program induction model is required to learn a program π that deduces the truth value tensor of the output predicate `HasSister`.

are the two current state-of-the-art NLMs[2] They introduce a neural-symbolic network architecture with strong inductive bias to realize the *forward chaining* (Evans & Grefenstette, 2018) reasoning mechanism. The *forward chaining* is a reasoning mechanism that conducts deduction based on the background atom sets to derive new atoms, then appends the newly derived atom to the atom set and sequentially repeats the process until the desired target output atom is derived. The concrete model architecture of the two models is shown in Figure 2. It is composed of $B \times D$ basic unit called *logic module* (LM), where B denotes the maximum breadth and D denotes the maximum depth (number of layers) of the model. A LM at breath $b \in \{0, \dots, B\}$ and layer $d \in \{0, \dots, D\}$ takes the truth value tensor of b-ary, $b-1$-ary and $b+1$-ary atoms from the previous layer as input: $I_b^d = \{\mathbf{p}_{b-1}^{d-1}, \mathbf{p}_b^{d-1}, \mathbf{p}_{b+1}^{d-1}\}$. Then the LM conducts input predicate combination and deduction to output newly invented atom $\mathbf{p}_b^d$: $\mathbf{p}_b^d \leftarrow \mathbf{p}_{b-1}^{d-1} \odot \mathbf{p}_b^{d-1} \odot \mathbf{p}_{b+1}^{d-1}$, $\odot$ denotes logical operators (i.e., conjunction, disjunction). Concretely, NLM uses a multi-layer perceptron (MLP) for the realization of the LM: $\mathbf{p}_b^d = \sigma(\mathrm{MLP}(\mathbf{p}_{b-1}^{d-1}, \mathbf{p}_b^{d-1}, \mathbf{p}_{b+1}^{d-1}))$. To improve the interpretability of the model, DLM proposes using fuzzy-logic operators (i.e., fuzzy conjunction, fuzzy disjunction) for the LM realization. The model sequentially applies the forward chaining to generate new predicates by stacking layers of multiple depth D to derive the desired target predicate.

Figure 3 shows an input & output representation of a reasoning task `HasSister`. The input background atoms include two unary atoms: $\texttt{HasFather}(x)$, $\texttt{HasMother}(x)$ and two binary atoms: $\texttt{IsMother}(x, y)$, $\texttt{IsDaughter}(x, y)$.

[2]The term *NLMs* used in this paper refers to general neural logic machine models, which include both NLM and DLM.

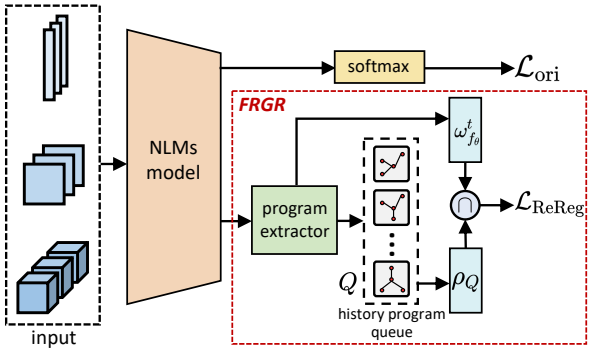

Figure 4. Overview of the FRGR framework.

For a family of four people: A, B, C, D: A, B are father and mother; C, D are the two daughters. The background atoms can be instantiated by replacing the variable with concrete family members (e.g., since B is C and D's mother, for the matrix of $\texttt{IsMother}(x, y)$: $\texttt{IsMother}(C, B) = 1$, $\texttt{IsMother}(D, B) = 1$). We therefore obtain the truth value tensor representation for each atom. For example, for the `IsMother` atom, the tensor representation is $\mathbf{p}^{\texttt{IsMother}} \in [0, 1]^{4\times4}$. Given the truth value table representation of the background atoms as input, The NLMs are required to learn the ground truth FOL program and generate the truth value tensor representation of the target output atom. *E.g.*, for the `HasSister` task, the ground truth FOL program to learn is: $\texttt{HasSister}(x) \leftarrow \exists y, z, \texttt{IsDaughter}(z, y) \wedge \texttt{IsMother}(x, z)$ and the NLMs should generate the truth value tensor representation of the target output atom $\texttt{HasSister}(x)$.

3. Methodology

Motivated by the above-mentioned intuition in Section 1, we now introduce our proposed method, called FRGR (Failure Reflection Guided Regularizer). The key idea of FRGR is to timely detect and summarize the root cause that is responsible for the NLM models' repetition of errors, therefore stopping it from oscillating around a local optimum by penalizing the root cause. The FRGR is applied dynamically as the learning proceeds. Figure 4 shows the overview of the FRGR framework. We first propose the *root cause mining* module that dynamically identifies the model's repetition of mistakes and summarizes the root cause subprogram (Section 3.1). Second, we introduce a novel regularization approach: *repetition regularization* that is used to avoid error repetition in future iterations based on the summarized root cause subprogram (Section 3.2).

3.1. Root Cause Mining

We first use an intuitive example of the learning process of a rule-based ILP model to illustrate the moti-

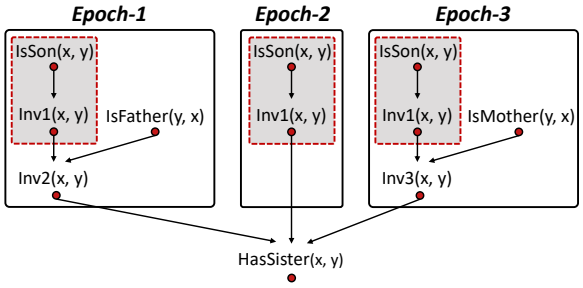

Figure 5. A motivating example of our approach.

vation of FRGR. The model is trying to synthesize a program for the `HasSister` task: $\texttt{HasSister}(x) \leftarrow \exists y, z, \texttt{IsDaughter}(z, y) \wedge \texttt{IsMother}(x, z)$. Figure 5 shows the induced programs of three successive iterations that are erroneous. It can be seen from the highlighted part of the figure that all three programs involve using the subprogram: $\texttt{Inv1}(x, y) \leftarrow \texttt{IsSon}(x, y)$. The model keeps updating the program by replacing the higher-level atoms (i.e., atoms closer to the target atom $\texttt{HasSister}(x, y)$ in the deduction process) while keeping the highlighted subprogram unchanged. It can be inferred based on human knowledge that as long as the induced program involves using this subprogram, it will always be erroneous or redundant. By identifying the model's repetitious usage of such root cause subprograms that are responsible for the erroneous induced program, we can leverage the root cause to avoid the model from repeating errors of such kind in future iterations and stop it from oscillating around an ill-performed local optimum.

Therefore, in order to identify whether the NLMs model is repeating mistakes of similar root causes, and summarize the root cause if it does during the training process, we propose a novel method called root cause mining (RCM), which contains three parts: (1) program extraction, (2) history program queue, (3) root cause mining.

3.1.1. Program Extractor

Let $f_{\boldsymbol{\theta}}^t$ be a NLMs model parameterized by weight tensor $\boldsymbol{\theta}$ at the training epoch t. Since the NLMs model does not contain an explicit logic program for deduction, a program extractor $\epsilon(\cdot)$ is utilized to extract the deduction process (i.e., program) from the neural model. Concretely, for NLM, for each MLP logic module $l \in f_{\boldsymbol{\theta}}^t$, we use the largest weight that connects to each output conclusive predicate (i.e., output neuron) as the approximation of its semantics. Finally, we collect the indices of all these weights (i.e., connections) of the logic module $S_l, l \in f_{\boldsymbol{\theta}}^t$ and we use the S_l of all logic modules as the representation of the induced program. Formally:

$$\begin{aligned}\omega_{f_{\boldsymbol{\theta}}}^t &= \epsilon({f_{\boldsymbol{\theta}}}^t) = \bigcup_{l \in f_{\boldsymbol{\theta}}^t} S_l \\ &= \bigcup_{l \in f_{\boldsymbol{\theta}}^t} \{(b_l, d_l, x, y) | 1 \le y \le m, x = \arg\max_x \mathrm{col}_y \boldsymbol{\theta}_l\}\end{aligned} \tag{1}$$

where $b_l \in \{1, \ldots, B\}, d_l \in \{1, \ldots, D\}$ denote the breadth and depth index of the logic module l, $\boldsymbol{\theta}_l \in \mathbb{R}^{m \times n}$ represents the weight matrix of logic module l, y is the index of an output neuron of l, and x denotes the index of the largest input neuron that connects to y: $x = \arg\max_x \mathrm{col}_y \boldsymbol{\theta}_l$.

Similarly, for DLM, the representation of the induced program is:

$$\begin{aligned}\omega_{f_{\boldsymbol{\theta}}}^t &= \epsilon({f_{\boldsymbol{\theta}}}^t) = \bigcup_{l \in f_{\boldsymbol{\theta}}^t} S_l \\ &= \bigcup_{l \in f_{\boldsymbol{\theta}}^t} \{(b_l, d_l, x, z, y) | 1 \le y \le m, x = \arg\max_x \mathrm{col}_{z,y} \boldsymbol{\theta}_l\},\end{aligned} \tag{2}$$

where $\boldsymbol{\theta}_l \in \mathbb{R}^{m \times 2 \times n}$ represents the weight matrix of logic module l, y is the index of an output neuron of l, $z \in \{1, 2\}$ denotes a sign that tells which fuzzy logic module the weight belongs to ($z = 1$ denotes the fuzzy conjunction and $z = 2$ represents the fuzzy disjunction), x is the index of the largest input neuron that connects to the output neuron: $\arg\max_x \mathrm{col}_{z,y} \boldsymbol{\theta}_l$.

3.1.2. History Program Queue

After obtaining the program representation of the NLMs model, we use a *history program queue* data structure to store it for the downstream root cause mining. Specifically, let $Q : |Q| = m$ be a *history program queue* of size m. During the training process of a model $f_{\boldsymbol{\theta}}^t$ at training step t, if the model's deduced conclusions on the mini-batch $u \in \mathcal{D}_{\text{train}}$ include error, we extract its current program representation ω_f^t and enqueue it into Q:

$$Q^{t+1} = \text{enqueue}(Q^t, \omega_{f_{\boldsymbol{\theta}}}^t), \exists (I, O) \in u : f_{\boldsymbol{\theta}}^t(I) \neq O \tag{3}$$

where $\text{enqueue}(Q, x)$ represents the operation of enqueuing element x into the queue Q, (I, O) is the input and output data of a sample.

3.1.3. Root Cause Mining

Given the history program queue Q of erroneous program representations, we now summarize the root cause subprogram that is responsible for error repetition if it happens. To

Algorithm 1 FRGR framework.

Input: maximum iteration step T, NLMs model $f_{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta}$, regulatory coefficient β

repeat
  for $t = 1, \ldots, T$ **do**
    Train $f_{\boldsymbol{\theta}}^t$ on mini-batch $(\mathbf{x}_t, \mathbf{y}_t) \sim \mathcal{D}_{\text{train}}$
    if $f_{\boldsymbol{\theta}}^t(\mathbf{x}_t) \neq \mathbf{y}_t$ **then**
      Extract program rep. and enqueue:
      $Q^{t+1} = \text{enqueue}(Q^t, \omega_f^t)$ ▷ Eq.1,2,3
    end
    if $|\, Q \,|| \, t$ **then**
      update root cause rep. with Apriori ▷ Eq.4
    end
    Update $f_{\boldsymbol{\theta}}$ with vanilla loss and ReReg loss:
    $\mathcal{L}_{\boldsymbol{\theta}} = \mathcal{L}_{\text{CLS}} + \beta \mathcal{L}_{\text{ReReg}}$ ▷ Eq.5,6
  end
until *reaching maximum training steps;*

approximate the root cause subprogram, we introduce an efficient realization called *root cause mining* (RCM). Concretely, we approximate the root cause with the frequently coexisting set of neurons among the collected erroneous program representations in Q. In specific, we adopt the Apriori algorithm (Agrawal & Srikant, 1994) and perform frequent item mining on Q to obtain the root cause subprogram representation ρ_Q, which is a set that contains the indices of the frequently coexisted neurons among the collected program representations $\omega_{f_{\boldsymbol{\theta}}}^t$. Formally,

$$\rho_Q = \left\{ J \subseteq S_{f_{\boldsymbol{\theta}}} \mid | \left\{ \omega_{f_{\boldsymbol{\theta}}}^t \in Q \mid J \subseteq \omega_{f_{\boldsymbol{\theta}}}^t \right\} | > \text{minsup} \right\} \tag{4}$$

where S_{f_θ} denotes the set that contains the indices of all weights of the NLMs model $f_{\boldsymbol{\theta}}$, J denotes a subset of neuron indices that coexisted in more than minsup many programs: $| \left\{ \omega_{f_{\boldsymbol{\theta}}}^t \in Q \mid J \subseteq \omega_{f_{\boldsymbol{\theta}}}^t \right\} | > \text{minsup}$, minsup is a minimum support threshold. During the training process, if the NLMs model repeats errors of similar root cause, the RCM module would promptly identify it by returning the summarized root cause ρ_Q. We perform RCM every m epochs to balance to overhead brought by the frequent itemset mining.

3.2. Repetition Regularization

With the summarized root cause subprogram ρ_Q obtained in the previous section, we now aim to utilize it to regularize the model and avoid it from repeating similar mistakes in future training. In particular, we propose a regularization term called *repetition regularization* (ReReg). ReReg measures the degree of repetition ϕ by taking the intersection of the current program representation of the model $\omega_{f_{\boldsymbol{\theta}}}^t$ and the root cause subprogram representation ρ_Q. Then we penalize the repetition with L1 regularization:

$$\begin{aligned} \mathcal{L}_{\text{ReReg}}(\boldsymbol{\theta}) &= \sum_{j=0}^{|\phi|} \|\phi_j\|_1 , \\ \phi &= \left\{ \boldsymbol{\theta}_{(b,d,x,y)} : (b,d,x,y) \in \omega_{f_{\boldsymbol{\theta}}}^t \cap \rho_Q \right\} \end{aligned} \tag{5}$$

Finally, the overall training objective function of FRGR contains the original conclusion classification loss and the ReReg loss $\mathcal{L}_{\text{ReReg}}$ as:

$$\mathcal{L}(\boldsymbol{\theta}) = \overbrace{-\sum_{i=0}^{|D|} y_i \cdot \log f_{\boldsymbol{\theta}}(x_i)}^{\text{conclusion classification}} + \overbrace{\beta \mathcal{L}_{\text{ReReg}}}^{\text{repetition regularization}} \tag{6}$$

where β is a regulatory coefficient. The detailed pseudo-code of the FRGR framework is shown in Algorithm 1.

4. Experiments

To demonstrate the effectiveness of FRGR, we evaluate the models under two settings: (1) high-resource data-rich setting, and (2) low-resource data-scare setting. For the following of this section, we answer three research questions (RQs) to lead our discussion:

(RQ1) Motivation Validation & Repetition Mitigation How serious is NLMs' repetition of erroneous subprograms? Is FRGR effective in mitigating the model's repetitious usage of erroneous subprograms?

(RQ2) Data-rich Setting Can FRGR improve the NLMs models under the data-rich setting?

(RQ3) Data-scarce Setting Can FRGR improve the NLMs models under the data-scarce setting?

4.1. Experimental Setup

Datasets. We follow previous work (Dong et al., 2018; Zimmer et al., 2023) and evaluate our framework on two reasoning benchmarks: relational reasoning and reinforcement learning[3]:

- **Relational reasoning.** The relational reasoning tasks contain two major categories: Family Tree Reasoning (Dong et al., 2018; Evans & Grefenstette, 2018) and General Graph Reasoning (Graves et al., 2016; Dong et al., 2018; Zimmer et al., 2023). For the Family Tree Reasoning tasks, each sample is a family tree consisting of n family members. The goal of this task is to induce more complex relations of the family members based on some basic background relation atoms: e.g., inducing the $\texttt{IsMGUnle}(x, y)$ atom (i.e., whether

[3]Please refer to the appendix for the detailed benchmark descriptions and the training setup.

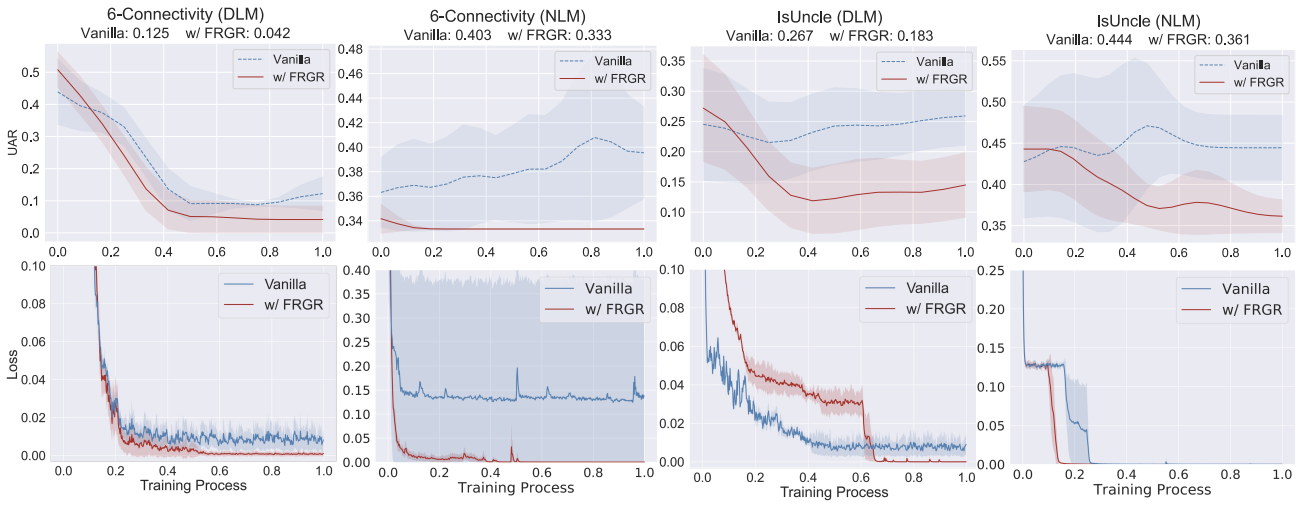

Figure 6. UAR during the training process (first row) and the training loss curve (second row). Blue lines represent training with vanilla classification loss only, and red lines represent training with FRGR regularization.

y is x's maternal great uncle) or `HasSister`(x) (i.e., whether x has at least one sister) based on the `IsMother`(x, y), `IsFather`(x, y), `IsSon`(x, y), and `IsDaughter`(x, y) background atoms. For the General Graph Reasoning tasks, each sample is an undirected graph sample that consists of n nodes. The background atom is `HasEdge`(x, y), which describes whether there exists an edge between node x and y. Based on the background atom, the goal of this benchmark is to induce complex properties of a node or relations between nodes: e.g., learning a logic program that determines whether two nodes can be connected by a path within k edges (k-`Connectivity`); or whether the out-degree of a node equals to k (k-`OutDegree`). To evaluate the model's performance and generalization, for the Family Tree Reasoning task, all the models are trained on family trees with 20 family members and tested on samples of family sizes of 20 (performance) and 100 (generalization). For the General Graph Reasoning task, all the models are trained on graphs with 10 nodes and tested on graphs with 10 nodes (performance) and 20 nodes (generalization).

- **Reinforcement learning.** We evaluate FRGR on three RL tasks: Sorting, Path (Graves et al., 2016), and Blocks World (Nilsson, 1982; Gupta & Nau, 1992). For the Sorting task, an array of length m is used as input, and the goal is to learn the `swap` predicate (i.e., swapping two integers in the array) to sort the list in ascending order. The learning is based on the following background atoms: `SmallerIndex(x,y)`, `SameIndex(x,y)`, `LargerIndex(x,y)`, `SmallerNumber(x,y)`, `SameNumber(x,y)`, `LargerNumber(x,y)`. For the Path environment, given an undirected graph represented by the background atom `HasEdge(x,y)`, the goal is to find a path between the start node s and the end node e, which are represented by two unary predicates (`IsStart`, `IsEnd`). The Blocks World task includes two worlds: an initial world and a target world, both of which contain m objects ($m-1$ cubes and 1 ground). The goal is to learn how to move the objects to change the world from the initial setting to the target setting. Each object is represented by four characteristics: `world_id`, `cube_id`, `coordinate_x`, and `coordinate_y`. The binary relations of all the above four characteristics are given as input: `SmallerWorldID(x,y)`, `SameWorldID(x,y)`, `LargerWorldID(x,y)`, `SmallerCubeID(x,y)`, `SameCubeID(x,y)`, `LargerCubeID(x,y)`, `SmallerX(x,y)`, `SameX(x,y)`, `LargerX(x,y)`, `SmallerY(x,y)`, `SameY(x,y)`, `LargerY(x,y)`. For RL tasks, all the models are trained with curriculum learning (Bengio et al., 2009) via REINFORCE algorithm (Williams, 1992) using environments containing less than 12 objects and tested on 10 and 50 objects.

Backbone Models. We conduct experiments on two state-of-the-art NLMs, namely, neural logic machines (NLM (Dong et al., 2018)) and differentiable logic machines (DLM (Zimmer et al., 2023)). For a fair comparison, we use the official code of the original papers and strictly follow the network structure and training setup. Please refer to the appendix for more implementation details.

Evaluation Metrics We adopt the same evaluation metrics used in previous works (Dong et al., 2018; Zimmer et al., 2023). The *success rate* measures the ratio of samples

Table 1. The comparative results of original NLMs and Ours (with FRGR) under the data-rich setting. n is the size of the data sample, e.g., how many members in a family. For RL tasks, all models are trained using curriculum learning on environments with $n \leq 12$. The number before the slash is the original NLMs' results, and after the slash, the NLMs w/ FRGR's result. Results in blue represent the original NLMs perform better, and red if NLMs w/ FRGR is better.

Task	NLM / **NLM w/ FRGR (Ours)**				DLM / **DLM w/ FRGR (Ours)**			
Family Tree	Grad-ratio (%)↑	n=20 (%)↑	n=100 (%)↑	# Epochs↓	Grad-ratio (%)↑	n=20 (%)↑	n=100 (%)↑	# Epochs↓
HasFather	100.00/100.00	100.00/100.00	100.00/100.00	5.90/6.00	100.00/100.00	100.00/100.00	100.00/100.00	22.00/23.6
HasSister	100.00/100.00	100.00/100.00	100.00/100.00	18.09/17.64	100.00/100.00	100.00/100.00	100.00/100.00	68.80/67.20
IsGrandparent	100.00/100.00	100.00/100.00	100.00/100.00	96.20/55.80	100.00/100.00	100.00/100.00	100.00/100.00	50.40/51.20
IsUncle	90.00/100.00	99.76/100.00	82.60/100.00	143.70/78.40	60.00/80.00	60.00/80.00	60.00/80.00	319.20/278.40
IsMGUncle	70.00/100.00	97.16/99.96	10.04/60.44	203.88/175.20	40.00/60.00	48.1/58.20	20.00/40.00	459.20/423.80
Graph Reasoning	Grad-ratio (%)↑	n=10 (%)↑	n=20 (%)↑	# Epochs↓	Grad-ratio (%)↑	n=10 (%)↑	n=20 (%)↑	# Epochs↓
1-OutDegree	100.00/100.00	100.00/100.00	100.00/100.00	14.30/17.00	100.00/100.00	100.00/100.00	100.00/100.00	46.20/50.00
2-OutDegree	90.00/100.00	96.52/100.00	90.80/100.00	77.9/13.40	100.00/100.00	100.00/100.00	100.00/100.00	81.60/73.60
4-Connectivity	100.00/100.00	100.00/100.00	100.00/100.00	16.80/20.50	100.00/100.00	100.00/100.00	100.00/100.00	90.40/87.40
6-Connectivity	60.00/100.00	74.40/100.00	69.20/100.00	278.00/41.60	80.00/80.00	86.90/95.40	53.28/90.10	282.40/230.80
Reinforcement Learning	Grad-ratio (%)↑	n=10 (%)↑	n=50 (%)↑	# Epochs↓	Grad-ratio (%)↑	n=10 (%)↑	n=50 (%)↑	# Epochs↓
Sorting	100.00/100.00	100.00/100.00	100.00/100.00	24.00/22.20	-	-	-	-
Path	50.00/60.00	99.55/100.00	99.95/100.00	311.00/305.20	-	-	-	-
BlocksWorld	40.00/60.00	97.11/96.59	76.89/83.90	390.11/386.67	-	-	-	-

in the data set that the models achieve 100% accuracy of a task. It is used to evaluate the model's performance and generalization. *Graduation ratio* measures the percentage of the training instances of different seeds that reach a success rate of 100% on the training set. Finally, *Epochs* measures the number of training epochs required to reach optimal success rate on the validation set. We use it to measure training efficiency.

4.2. Results Analysis

4.2.1. Motivation Validation & Repetition Mitigation (RQ1)

First, we show how serious NLMs' repetition of erroneous subprograms is and therefore validate the motivation of FRGR. Specifically, we analyze a specific kind of erroneous subprogram and experiment on two tasks: `6-Connectivity` and `IsUncle`. The ground-truth programs of the two evaluated tasks do not contain any unary predicates (refer to the appendix for detailed ground-truth programs), thus it is erroneous if the extracted program representation $\omega^t_{f_\theta}$ of the NLMs contains a high proportion of unary predicates. To quantify the evaluation, we use a measurement called *unary atom ratio* (UAR), which computes the ratio of unary atoms in $\omega^t_{f_\theta}$. We visualize the UAR value during the training process, the results are shown in the first row of Figure 6. We observe that during the training process, the UAR of training with vanilla loss only is high and often fails to decrease. This demonstrates the motivation that the NLMs are repeating such mistakes. Training with FRGR manages to decrease the UAR effectively. We further present the UAR of the converged models (shown in the title of each figure), the UAR decreases significantly with the use of FRGR.

We further investigate whether FRGR is effective in optimizing the learning process by decreasing the repetition of mistakes. We visualize the training loss curves of different methods as shown in the second row of Figure 6. We have two key observations: (1) we observe that FRGR can help achieve a much well-performed optimum with lower training loss compared with training with vanilla loss function only (2) when oscillation around a local optimum occurs, FRGR can timely stop it and continue the loss decreasing.

In summary, the experiments show that the NLMs would repeat similar erroneous subprograms, and FRGR can effectively reduce the repetition and optimize the learning process.

4.2.2. Data-rich Setting (RQ2)

We first follow the setup of previous works (Dong et al., 2018; Zimmer et al., 2023) and conduct experiments under the ideal data-rich setting (i.e., the number of training samples are sufficient). The results are shown in Table 1. In particular, from top to bottom, we range the tasks from the simplest to the hardest according to the number of predicates involved in their ground-truth programs.

We observe that FRGR can significantly improve the NLMs' test performance and generalization. *E.g.*, For the most difficult Family Tree reasoning task: IsMGUncle, NLM w/ FRGR achieves near-optimal IID performance and five times higher generalization over NLM. Besides, FRGR can also improve the training process of the NLMs by achieving a much better graduation ratio on both relational reasoning and reinforcement learning tasks. For the training efficiency, we observe that with the introduction of FRGR, the models require much fewer epochs to converge. *E.g.*, for the

Table 2. The comparative results of original NLMs and Ours (with FRGR) under the data-scarce setting. The training settings are the same as the data-rich setting except for the number of training data. The number before the slash is the original NLMs' results, and after the slash, the NLMs w/ FRGR's result. Results in blue represent the original NLMs perform better, and red if NLMs w/ FRGR is better.

Task	NLM / **NLM w/ FRGR (Ours)**				DLM / **DLM w/ FRGR (Ours)**			
Family Tree	Grad-ratio (%)↑	n=20 (%)↑	n=100 (%)↑	# Epochs↓	Grad-ratio (%)↑	n=20 (%)↑	n=100 (%)↑	# Epochs↓
HasFather	100.00/100.00	100.00/100.00	100.00/100.00	5.50/ 5.50	100.00/100.00	100.00/100.00	100.00/100.00	23.20/27.00
HasSister	100.00/100.00	100.00/100.00	100.00/100.00	13.20/13.30	100.00/100.00	100.00/100.00	100.00/100.00	67.20/68.00
IsGrandparent	90.00/100.00	68.70/77.51	63.40/72.26	127.90/54.30	100.00/100.00	100.00/100.00	100.00/100.00	57.14/56.00
IsUncle	100.00/100.00	96.49/98.05	62.53/81.50	134.30/102.80	40.00/80.00	40.20/80.00	40.00/80.00	401.60/362.80
IsMGUncle	70.00/100.00	68.52/94.20	33.00/48.00	356.30/251.60	0.00/0.00	0.00/0.00	0.00/0.00	500.00/500.00
Graph Reasoning	Grad-ratio (%)↑	n=10 (%)↑	n=20 (%)↑	# Epochs↓	Grad-ratio (%)↑	n=10 (%)↑	n=20 (%)↑	# Epochs↓
1-OutDegree	100.00/100.00	100.00/100.00	100.00/100.00	52.90/61.10	100.00/100.00	100.00/100.00	100.00/100.00	47.20/48.00
2-OutDegree	90.00/100.00	99.92/100.00	99.72/100.00	135.70/73.30	100.00/100.00	100.00/100.00	100.00/100.00	92.00/83.62
4-Connectivity	100.00/100.00	100.00/100.00	100.00/100.00	151.60/195.10	100.00/100.00	100.00/100.00	100.00/100.00	82.80/68.00
6-Connectivity	80.00/80.00	63.20/77.20	39.40/70.80	63.75/38.25	20.00/40.00	75.30/86.30	59.80/70.00	424.00/359.20
Reinforcement Learning	Grad-ratio (%)↑	n=10 (%)↑	n=50 (%)↑	# Epochs↓	Grad-ratio (%)↑	n=10 (%)↑	n=50 (%)↑	# Epochs↓
Sorting	100.00/100.00	100.00/100.00	100.00/100.00	28.20/24.60	-	-	-	-
Path	70.00/100.00	98.94/99.88	93.65/99.80	304.60/206.00	-	-	-	-
BlocksWorld	40.00/40.00	84.13/90.13	45.93/52.60	414.00/442.00	-	-	-	-

`IsUncle` task, NLM w/ FRGR requires 45.44% fewer epochs than NLM to converge to the optimal solution; DLM w/ FRGR requires 12.70% fewer epochs than DLM. Similarly, for the reinforcement learning benchmarks, NLM w/ FRGR manages to reduce the number of epochs for all the evaluated tasks[4]. We notice that the numbers of epochs required for the `HasFather`, `1-OutDegree`, and `4-Connectivity` tasks are slightly increased. After our investigation, we attribute this to the fact that these tasks are rather straightforward to learn (less than 20 epochs are required for NLM to converge to the optimal solutions). While FRGR introduces additional regularization to the learning process, which may result in a slower learning speed but works better on more challenging tasks that require complex logical reasoning.

4.2.3. Data-scarce Setting (RQ3)

To address RQ3, we simulate the data-scare scenario by using only $1/500$ of the data-rich training data volume on the relational reasoning benchmark. The results are shown in Table 2. The experiment illustrates that the performance and generalization of the original NLMs decrease considerably compared with the data-rich results, which demonstrate the data-hungry nature of the NLMs. With the usage of FRGR, we observe a significant improvement in terms of all evaluation metrics. *E.g.*, on the `IsUncle` task, the DLM's performance and generalization drops by 20% (i.e., 60.00% to 40.00%), while DLM w/ FRGR manages to keep the same performance and generalization under the data-rich setting (i.e., 80.00%). The results demonstrate that FRGR is effective in improving the NLMs' data efficiency by boosting its results under the data-scare setting.

[4]We have contacted the DLM's authors yet we are unable to reproduce the results of the reinforcement learning benchmarks reported in the original DLM paper

5. Related Work

Neural Program Induction & Synthesis. Program induction and synthesis are the tasks that aim to learn programs that satisfy pre-defined program specifications. In recent years, with the development of deep learning, neural networks-based program induction and synthesis have been proven effective in logical reasoning tasks with much less manual design effort (Evans & Grefenstette, 2018; Devlin et al., 2017; Chen et al., 2018; Bunel et al., 2018). Evans et al. (Evans & Grefenstette, 2018) proposes a differentiable implementation of inductive logic programming (∂ILP) which is capable of synthesizing white-box Datalog programs given noisy input data. Based on ∂ILP, Cao et al. (Cao et al., 2022) propose a sketch-based program synthesis framework for reinforcement learning tasks and achieves high performance, generalizability, and knowledge reusability. Neural Logic Machines (NLM) (Dong et al., 2018) proposes using a novel rule induction system that implements boolean logic rules and quantifications with neural modules. Trivedi et al. (Trivedi et al., 2021) propose a neural program synthesis framework that first learns a program embedding space that parameterizes behaviors in an unsupervised manner, then generates a program that maximizes the return of a task by searching over the program embedding space. The proposed framework manages to outperform previous deep reinforcement learning and program synthesis baselines.

Relational Inductive Bias. Deep learning models easily suffer from bias issues (Tommasi et al., 2017; Du et al., 2021; Li et al., 2022; 2023a;b). However, relational inductive biases of neural network architectures can improve

models' learning about entities and relations. For example, the relational inductive biases of graph networks can improve combinatorial generalization and sample efficiency (Battaglia et al., 2018). For reinforcement learning, architectural inductive biases within a deep reinforcement learning agent are effective in learning relations (Zambaldi et al., 2018). Besides, the architectural inductive bias also contributes to the good performance of the inductive logic programming models (Dong et al., 2018; Zimmer et al., 2023).

6. Conclusions

In this work, we propose a novel regularization framework called FRGR, which improves the optimization of the NLMs models by utilizing the root cause of error repetition. Our proposed method first summarizes the root cause of errors from the models' previous behavior with pattern mining techniques. Then based on the summarized root cause, FRGR penalizes the model if it repeats similar mistakes in future training iterations. Experimental results on multiple reasoning benchmarks demonstrate that FRGR can effectively improve the NLMs' performance, generalization, training efficiency, and data efficiency. In future work, we would like to further develop FRGR by applying the idea to other fields and model architectures.

Acknowledgement

This research is supported by the National Research Foundation, Singapore, and the Cyber Security Agency under its National Cybersecurity R&D Programme (NCRP25-P04-TAICeN) and NRF Investigatorship NRF-NRFI06-2020-0001, the National Natural Science Foundation of China (Grant No. 62106172), the Science and Technology on Information Systems Engineering Laboratory (Grant Nos. WDZC20235250409, 6142101220304), the Xiaomi Young Talents Program of Xiaomi Foundation, and the RIE2020 Industry Alignment Fund – Industry Collaboration Projects (IAF-ICP) Funding Initiative, as well as cash and in-kind contributions from the industry partner(s). Any opinions, findings and conclusions, or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore, and Cyber Security Agency of Singapore.

Impact Statement This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Agrawal, R. S. and Srikant, R. R. fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, pp. 487–499, 1994.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

Bunel, R., Hausknecht, M., Devlin, J., Singh, R., and Kohli, P. Leveraging grammar and reinforcement learning for neural program synthesis. In *International Conference on Learning Representations*, 2018.

Cao, Y., Li, Z., Yang, T., Zhang, H., Zheng, Y., Li, Y., Hao, J., and Liu, Y. Galois: boosting deep reinforcement learning via generalizable logic synthesis. *Advances in Neural Information Processing Systems*, 35:19930–19943, 2022.

Chen, X., Liu, C., and Song, D. Execution-guided neural program synthesis. In *International Conference on Learning Representations*, 2018.

Devlin, J., Uesato, J., Bhupatiraju, S., Singh, R., Mohamed, A.-r., and Kohli, P. Robustfill: Neural program learning under noisy i/o. In *International conference on machine learning*, pp. 990–998. PMLR, 2017.

Dong, H., Mao, J., Lin, T., Wang, C., Li, L., and Zhou, D. Neural logic machines. In *International Conference on Learning Representations*, 2018.

Du, M., Manjunatha, V., Jain, R., Deshpande, R., Dernoncourt, F., Gu, J., Sun, T., and Hu, X. Towards interpreting and mitigating shortcut learning behavior of nlu models. *arXiv preprint arXiv:2103.06922*, 2021.

Evans, R. and Grefenstette, E. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

Garcez, A. d., Bader, S., Bowman, H., Lamb, L. C., de Penning, L., Illuminoo, B., Poon, H., and Zaverucha, C. G. Neural-symbolic learning and reasoning: A survey and interpretation. *Neuro-Symbolic Artificial Intelligence: The State of the Art*, 342(1):327, 2022.

Getoor, L. and Taskar, B. *Introduction to statistical relational learning*. MIT press, 2007.

Getoor, L., Friedman, N., Koller, D., and Pfeffer, A. Learning probabilistic relational models. *Relational data mining*, pp. 307–335, 2001.

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

Gupta, N. and Nau, D. S. On the complexity of blocks-world planning. *Artificial Intelligence*, 56(2-3):223–254, 1992.

Horn, A. On sentences which are true of direct unions of algebras1. *The Journal of Symbolic Logic*, 16(1):14–21, 1951.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. Building machines that learn and think like people. *Behavioral and brain sciences*, 40:e253, 2017.

Li, T., Guo, Q., Liu, A., Du, M., Li, Z., and Liu, Y. Fairer: fairness as decision rationale alignment. In *International Conference on Machine Learning*, pp. 19471–19489. PMLR, 2023a.

Li, T., Li, Z., Li, A., Du, M., Liu, A., Guo, Q., Meng, G., and Liu, Y. Fairness via group contribution matching. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, pp. 436–445, 2023b.

Li, Z., Li, Y., Li, T., Du, M., Wu, B., Cao, Y., Jiang, J., and Liu, Y. Unveiling project-specific bias in neural code models. *arXiv preprint arXiv:2201.07381*, 2022.

Li, Z., Cao, Y., Xu, X., Jiang, J., Liu, X., Teo, Y. S., Lin, S.-w., and Liu, Y. Llms for relational reasoning: How far are we? *arXiv preprint arXiv:2401.09042*, 2024.

Muggleton, S. Inductive logic programming. *New generation computing*, 8:295–318, 1991.

Muggleton, S. H., Lin, D., and Tamaddoni-Nezhad, A. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *Machine Learning*, 100(1):49–73, 2015.

Nilsson, N. J. *Principles of artificial intelligence*. Springer Science & Business Media, 1982.

Raghothaman, M., Mendelson, J., Zhao, D., Naik, M., and Scholz, B. Provenance-guided synthesis of datalog programs. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–27, 2019.

Susskind, Z., Arden, B., John, L. K., Stockton, P., and John, E. B. Neuro-symbolic ai: An emerging class of ai workloads and their characterization. *arXiv preprint arXiv:2109.06133*, 2021.

Tommasi, T., Patricia, N., Caputo, B., and Tuytelaars, T. A deeper look at dataset bias. *Domain adaptation in computer vision applications*, pp. 37–55, 2017.

Trivedi, D., Zhang, J., Sun, S.-H., and Lim, J. J. Learning to synthesize programs as interpretable and generalizable policies. *Advances in neural information processing systems*, 34:25146–25163, 2021.

Valmeekam, K., Marquez, M., Olmo, A., Sreedharan, S., and Kambhampati, S. Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pp. 5–32, 1992.

Zambaldi, V., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D., Lillicrap, T., Lockhart, E., et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

Zimmer, M., Feng, X., Glanois, C., JIANG, Z., Zhang, J., Weng, P., Li, D., HAO, J., and Liu, W. Differentiable logic machines. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=mXfkKtu5JA.

A. Benchmark Details

In this section, we illustrate the details of the reasoning benchmarks.

Family tree reasoning. The family tree reasoning benchmark consists of tasks that require the model to induce programs that deduce more complex relations based on some basic properties of family members or relations between them. Specifically, a family tree is represented with four basic predicates: $\texttt{IsMother}(x, y)$, $\texttt{IsSon}(x, y)$, $\texttt{IsSon}(x, y)$, $\texttt{IsDaughter}(x, y)$. *E.g.*, $\texttt{IsMother}(x, y)$ is `True` if y is x's mother, the semantics of the other basic predicates are similar. This benchmark contains 5 target predicates to induce. The details are as follows:

- $\texttt{HasFather}(x)$: the semantics of $\texttt{HasFather}(x)$ is to determine whether x has a father. The ground-truth program to induce is:

$$\texttt{HasFather}\ (x) \leftarrow \exists y,\ \texttt{IsFather}\ (x, y) \quad (7)$$

- $\texttt{HasSister}(x)$: the semantics of this predicate is to determine whether x has a sister. The ground-truth program to induce is:

$$\texttt{HasSister}(x) \leftarrow \exists y, z, \texttt{IsDaughter}(z, y) \wedge \texttt{IsMother}(x, z) \quad (8)$$

- $\texttt{IsGrandparent}(x, y)$: the semantics of this predicate is to determine whether y is the grandparent of x. The ground-truth program to induce is:

$$\texttt{IsGrandparent}(x, y) \leftarrow \exists z, ((\texttt{IsSon}(y, z) \wedge \texttt{IsFather}(x, z)) \vee (\texttt{IsDaughter}(y, z) \wedge \texttt{IsMother}(x, z))) \quad (9)$$

- $\texttt{IsUncle}(x, y)$: the semantics of this predicate is to determine if y is the uncle of x. The ground-truth program to induce is:

$$\begin{aligned}&\texttt{IsUncle}(x, y) \leftarrow \exists z, ((\ \texttt{IsMother}\ (x, z) \wedge \texttt{Invented}(z, y))) \vee (\ \texttt{IsFather}\ (x, z) \wedge \texttt{Invented}(z, y))\\ &\texttt{Invented}(x, y) \leftarrow \exists z, ((\texttt{IsSon}(z, y) \wedge \texttt{IsSon}(z, x)) \vee (\texttt{IsSon}(z, y) \wedge \texttt{IsDaughter}(z, x)))\end{aligned} \quad (10)$$

- $\texttt{IsMGUncle}(x, y)$: the semantics of this predicate is to determine whether y is the maternal great uncle of x. The ground-truth program to induce is:

$$\texttt{IsMGUncle}(x, y) \leftarrow \exists z, (\texttt{IsMother}(x, z) \wedge \texttt{IsUncle}(z, y)) \quad (11)$$

General graph reasoning. The general graph reasoning benchmark consists of tasks that require the models to infer the logic of high-level target predicates that describe properties/relations of a graph based on a basic predicate: $\texttt{HasEdge}(x, y)$ (i.e., whether there is an undirected edge between node x and y in the graph). This benchmark contains 4 target predicates to infer. The details are as follows:

- $\texttt{4-Connectivity}(x, y)$: the semantics of $\texttt{4-Connectivity}(x, y)$ is to determine whether there exists a path between node x and node y within 4 edges. The ground-truth program to induce is:

$$\begin{aligned}&\texttt{4-Connectivity}\ (x, y) \leftarrow \exists z, (\ \texttt{HasEdge}\ (x, y) \vee\ \texttt{Invented}\ (x, y) \vee (\ \texttt{Invented}\ (x, z) \wedge \texttt{HasEdge}(z, y)) \vee (\ \texttt{Invented}\ (x, z) \wedge\ \texttt{Invented}\ (z, y)))\\ &\texttt{Invented}(x, y) \leftarrow \exists z, (\texttt{HasEdge}(x, z) \wedge \texttt{HasEdge}(z, y))\end{aligned} \quad (12)$$

- $\texttt{6-Connectivity}(x, y)$: the semantics of $\texttt{6-Connectivity}(x, y)$ is to determine whether there exists a path between node x and node y within 6 edges. The ground-truth program to induce is:

$$\begin{aligned}&\texttt{6-Connectivity}\ (x, y) \leftarrow \exists z, (\ \texttt{HasEdge}\ (x, y) \vee\ \texttt{Invented1}\ (x, y) \vee \texttt{Invented2}\ (x, y) \vee (\texttt{Invented1}(x, z) \wedge \texttt{Invented1}(z, y))\\ &\vee (\ \texttt{Invented2}\ (x, z) \wedge \texttt{Invented1}\ (z, y)) \vee (\texttt{Invented2}(x, z) \wedge \texttt{Invented2}\ (z, y)))\texttt{Invented1}(x, y) \leftarrow \exists z, (\texttt{HasEdge}\ (x, z) \wedge \texttt{HasEdge}(z, y))\texttt{Invented2}(x, y)\\ &\leftarrow \exists z, (\texttt{HasEdge}(x, z) \wedge \texttt{Invented1}(z, y))\end{aligned} \quad (13)$$

- $\texttt{1-Outdegree}(x)$: the semantics of this predicate is to determine whether the outdegree of node x in a graph is exactly 1. The ground-truth program to induce is:

$$\texttt{1-Outdegree}\ (x) \leftarrow \exists y, \forall z, (\texttt{HasEdge}(x, y) \wedge \neg\texttt{HasEdge}(x, z)) \quad (14)$$

- 2-Outdegree(x): the semantics of this predicate is to determine whether the outdegree of node x in a graph is exactly 2. The ground-truth program to induce is:

$$\texttt{2-Outdegree}\ (x) \leftarrow \exists z, w, \forall y (\neg\texttt{HasEdge}(x, y) \wedge \texttt{HasEdge}(x, z) \wedge \texttt{HasEdge}(x, w)) \quad (15)$$

Algorithm 1 FRGR framework for reinforcement learning tasks.

Input: maximum number of episodes Epi, maximum iteration step T, NLMs model $f_{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta}$, regulatory coefficient β

1 **repeat**
2 Running Policy π_θ for T steps
3 Collecting trajectory $Traj = \{(s_t, a_t, r_t, \omega_t)\}_{0...T}$
4 Calculating the discounted return $\{G_1, G_2, \ldots, G_T\}$
5 **if** $Traj$ *Fails* **then**
6 Update Q according to $\{(G_t, \omega_t)\}_{0...T}$ ▷ Eq.1, 3
7 **end**
8 **if** $|\ Q\ ||\ t$ **then**
9 update root cause rep. with Apriori ▷ Eq.4
10 **end**
11 Update $f_{\boldsymbol{\theta}}$ with REINFORCE loss and ReReg loss:
12 $\mathcal{L}_{\boldsymbol{\theta}} = \mathcal{L}_{\text{REI}} - \beta\mathcal{L}_{\text{ReReg}}$ ▷ Eq.16
13 **until** *reaching maximum number of episodes;*

B. FRGR for the Reinforcement Learning Tasks

In this section, we illustrate the details of the FRGR framework for reinforcement learning tasks. Reinforcement learning tasks focus on making sequential decisions to complete the tasks with the goal of maximizing the returns (at time step t, the parameterized policy will take action a_t at state s_t, and continue until the task is completed or reaches the maximum steps allowed). Consequently, determining the correctness of a single action within an RL sequence is challenging. Therefore, after finishing a sequence, the final return of each state-action pair taken in that sequence will be calculated using a discounting factor. RL algorithms mainly optimize policies toward the direction of maximizing the returns. Similarly, in FRGR, the discounted returns are also used to determine which behavioral snapshots are used for error pattern mining. Algorithm 1 shows the detailed steps.

Given a reinforcement learning task, the model (policy π_θ) interacts with the environment for a maximum of T steps. For each step t, the model takes the grounded state s_t and outputs the action a_t. The program representation is also extracted (ω_t). Then, by interacting with the environment via a_t, the environment gives the reward r_t and next state s_{t+1}. After one episode, the discounted returns $(G_1, G_2, \ldots, G_T)$ are calculated based on the rewards collected, as shown from line 2 to 4. If the model fails to complete the task in this episode, we update the history program queue Q based on the discounted returns. Specifically, Q is implemented as a max heap with the size τ, storing the tuple (G_t, ω_t). For every new tuple (G_i, ω_i), when its return is smaller than the return of the root node, it is added to the history program queue Q, as shown in line 6. In this way, the subprograms that cause the lowest returns are considered the (most) root cause subprograms. Finally, similar to the relational reasoning scenario, the history program queue is used for root cause mining, as shown in line 9. The summarized root cause subprogram is then used as the regularization to penalize the final loss, as shown in line 11 and line 12.

C. Implementation Details

In this section, we illustrate the training details. We conduct all experiments on a Ubuntu 18.05 server with 48 cores of Intel Xeon Siver 4214 CPU, 4 NVIDIA Quadro RTX 8000 GPUs, and 252GB RAM.

Training Method. For our method, we strictly follow the training settings of both NLM (Dong et al., 2018) and DLM(Zimmer et al., 2023). They are trained using Adam optimizer (Kingma & Ba, 2014) with a 0.005 learning rate. For all the relational reasoning tasks, the Softmax Cross Entropy is used as the loss function. For reinforcement learning tasks, the REINFORCE (Williams, 1992) algorithm is used for optimization in NLM[5]. For NLM w/ FRGR, the policy entropy term is also added to the loss function. By adding the behavioral regularization term, parameters θ of the RL policy π is updated via:

$$\theta' = \theta + \eta[\overbrace{\gamma_r^t \nabla_\theta \log \pi_\theta Q_{\pi_\theta}(s_t, a_t) + \lambda \nabla_\theta H(\pi_\theta)}^{\text{REINFORCE LOSS}} - \beta \nabla \overbrace{\mathcal{L}_{\text{ReReg}}}^{\text{repetition regularization}}] \tag{16}$$

where η is the learning rate, γ_r^t is the discounted reward at time step t, H is the entropy regularization, λ is the discount factor to control the entropy, β is the regularization coefficient, s_t and a_t are the state and action at time step t. Across all the environments, a positive of $+1.00$ will be given to the agent. To encourage the agent to use as few moves as possible, a negative reward of -0.01 is given for each action taken.

Curriculum Learning. For reinforcement learning tasks, curriculum learning (Bengio et al., 2009; Dong et al., 2018) is also applied in NLM. The training instances are grouped into lessons according to their complexity. The number of objects in the environment is considered an indicator of complexity. The model will start with a simple lesson and gradually increase the difficulty when the model passes the exam. The exam will be taken when the model is well-trained on the current lesson, i.e., the accuracy reaches a certain threshold. Specifically, during the lesson, all failed and successful environments will be recorded. The training examples will be sampled from the successful environments with the probability of Ω and failed environments with the probability of $1 - \Omega$.

[5]Since we are unable to reproduce the results of DLM for RL tasks, we ignore the details for DLM on RL tasks.

Table 3. The details of the network structure for all the models for the relational reasoning tasks. Residual indicates the use of Input/Output residual links.

	Tasks	Depth NLM / DLM	Breadth	Residual	Examples (Data-rich)	Examples (Data-scarce)
Family Tree	HasFather	4/5	3	No	50,000	100
	HasSister	4/5	3	No	50,000	100
	IsGrandparent	4/5	3	No	100,000	200
	IsUncle	4/5	3	No	100,000	200
	IsMGUncle	4/9	3	No	200,000	400
General Graph	1-Outdegree	4/5	3	No	50,000	100
	2-Outdegree	5/7	4	Yes	100,000	200
	4-Connectivity	4/5	3	No	50,000	100
	6-Connectivity	8/9	3	Yes	50,000	100

Table 4. The details of the network structure and hyperparameters for the NLM and the NLM w/ FRGR models for the reinforcement learning tasks. The Lessons indicate the different levels of lessons used for training.

Tasks	Depth NLM	Breadth	Residual	Lessons	Ω	Epochs	Total Episodes (Data-rich)	Total Episodes (Data-scarce)
Sorting	3	2	Yes	[4,10]	0.5	50	1,000	140
Path	5	3	Yes	[3,12]	0.5	400	24,000	600
BlocksWorld	7	2	Yes	[2,12]	0.6	500	50,000	1,100

Hyperparameters for relational reasoning tasks. For our method, we keep all the hyperparameters the same as NLM and DLM. The details of the network structure of both NLM and DLM are shown in Table 3. For each computation unit, the number of intermediate predicates (hidden dimension) is set to be 8 for all the benchmarks. Specifically, the residual means the input predicates are concatenated to the output predicates of each computation unit. For the data-rich scenario, the examples are divided into 500 epochs, each containing different samples. For the data-scarce scenario, the examples are the same for each epoch. The batch size is set to be 4 across all the experiments. The regulatory coefficient β is set to be 0.1 and τ is set to be 100 for all tasks.

Hyperparameters for reinforcement learning tasks. Table 4 shows the details of the network structure and hyperparameters for reinforcement learning tasks. Each training batch contains one episode. Similarly, no hidden layer is used and the number of intermediate predicates is also set to be 8. Residual linkage is applied for all the RL tasks. Specifically, for curriculum learning, the NLM starts from a small number of objects and gradually advances to a larger number. For example, the first lesson for the Sorting task contains environments with 2 objects. The second lesson contains environments with 3 objects, and the final lesson contains environments with 10 objects. For the data-rich scenario, the environments are different for each lesson taken. For the data-scarce setting, the environments are the same for the same level of lessons. The regulatory coefficient β is set to be 0.1 and τ is set to be 100 for all tasks.

D. Hyperparameter Analysis

In this section, we present more experiments on the two introduced hyperparameters β and τ to understand FRGR better. More specifically, we conduct experiments on `IsUncle` task from Family Tree Reasoning tasks and `6-Connectivity` from Graph Reasoning tasks.

Analysis on β We conduct experiments with five different values of β $(0.1, 0.05, 0.01, 0.005, 0.001)$ for FRGR. The results in Figure 7 indicate that FRGR steadily improves over the original NLM for all evaluated settings of β. However, the training efficiency, performance, and generalization become sub-optimal when β becomes lower as the model with FRGR will gradually downgrade to the original model (when β equals zero). With a higher β, FRGR can more efficiently regularize the model by penalizing it for repeating errors.

Analysis on τ We conduct experiments with four different values of τ $(50, 100, 150, 200)$ for FRGR. The results are shown in Figure 8. The experiments indicate that with

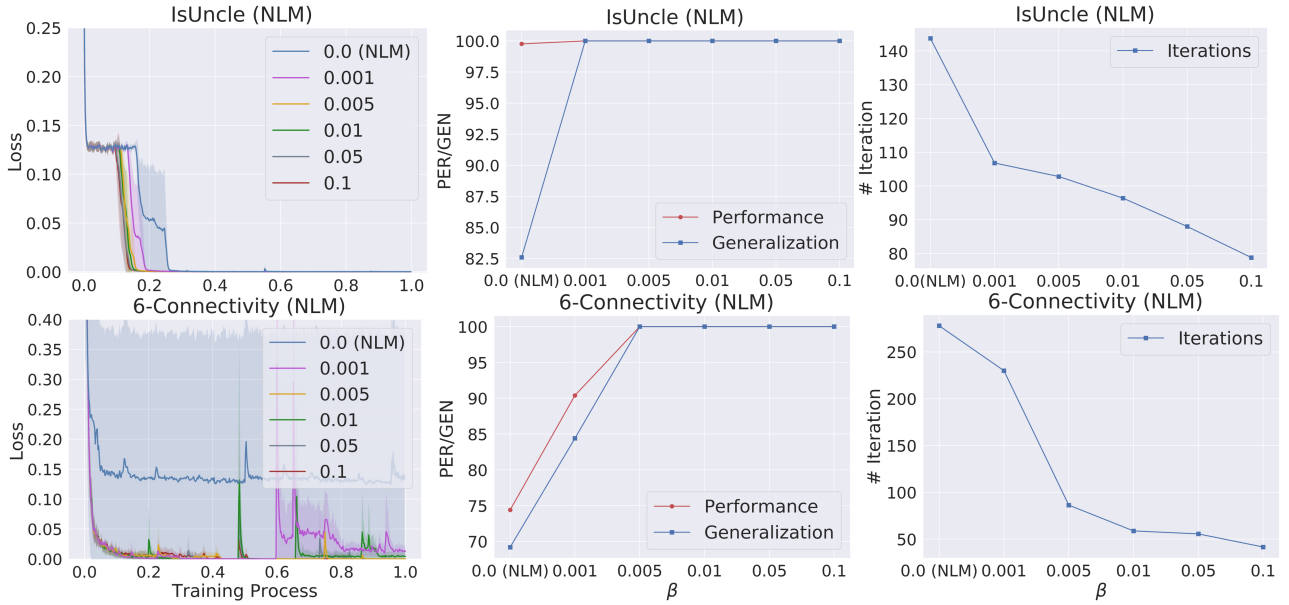

Figure 7. Experiments on the hyperparameter β to see its influence.

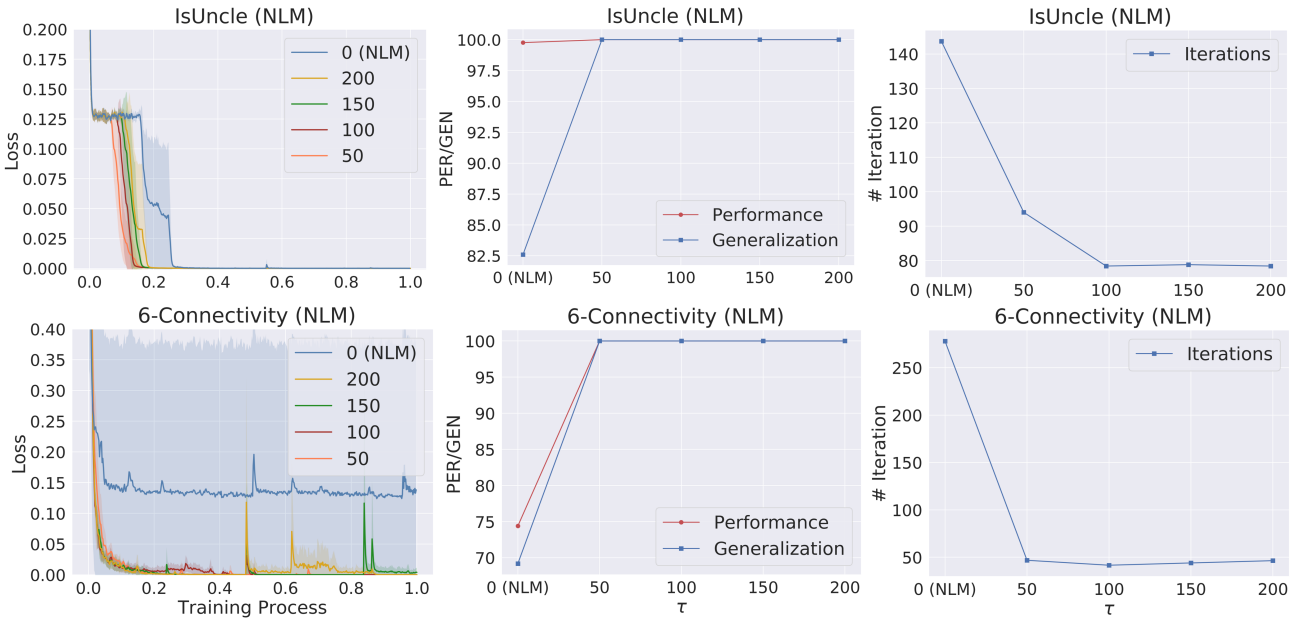

Figure 8. Experiments on the hyperparameter τ to see this influence.

different buffer sizes, FRGR can steadily improve over the original model. Under the evaluated values, varying the buffer size has little effect on performance, generalization, and training efficiency.

E. More Results of Motivation Validation & Repetition Mitigation

A high UAR during the training process indicates that the model relies heavily on the erroneous subprograms (unary atoms) for deduction which therefore results in a poor performance (high loss value). We plot the repetition regularization loss (green line) along all the other lines, as shown in Figures 9 and 10. We can observe from the results that when the classification loss of w/ FRGR is oscillating around

a local optima, the repetition regularization loss would significantly increase. With the regularization applied, the classification loss would start to decrease again and the model therefore escapes local optima. The results indicate that the repetition regularization loss is correlated with the original classification loss.

Additionally, we further investigate the tasks that require simple task-solving logic yet FRGR increases the number of training epochs required (i.e.,, the `HasFather`, 1-`OutDegree`, and 4-`Connectivity` tasks). Specifically, we follow Figure 6 and present the training loss curve and the unary atom ratio (UAR) (or ternary atom ratio (TAR)) during the training process. Note that for the 1-Outdegree task, we study the TAR, as the ground-truth program involves using unary & binary atoms, but no ternary atoms. We conduct the investigation on the NLM model. The results are shown in Figure 11. We can observe that though FRGR slightly increases the number of training epochs required, it still effectively decreases the UAR/TAR during the training process. In the meanwhile, though the vanilla NLM model achieves perfect performance & generalization (100%) on these tasks, its induced programs are redundant (i.e., high in UAR/TAR).

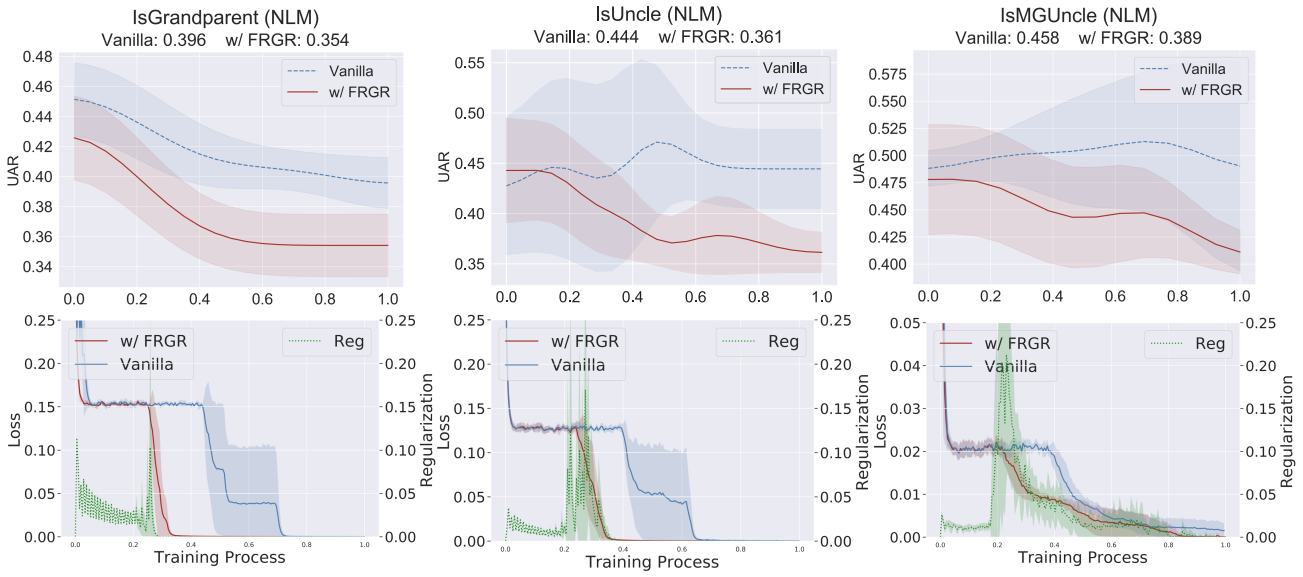

Figure 9. UAR during the training process (first row) and the training loss curve (second row) for IsGrandparent, IsUncle, and IsMGUncle tasks. Blue lines represent training with vanilla classification loss only, and red lines represent training with FRGR regularization. The greed dotted lines represent the magnitude of FRGR regularization.

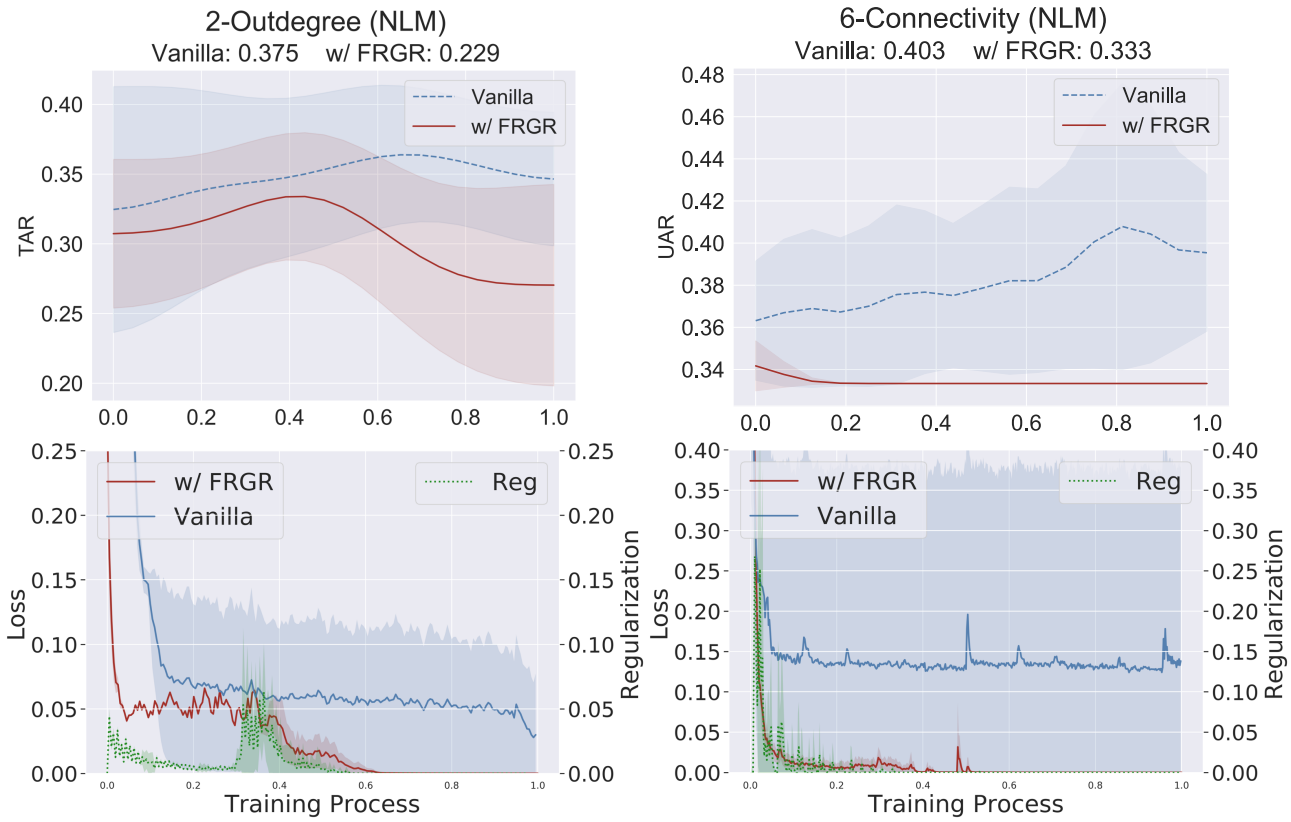

Figure 10. UAR during the training process (first row) and the training loss curve (second row) for 2-outdegree and 6-Connectivity tasks. Blue lines represent training with vanilla classification loss only, and red lines represent training with FRGR regularization. The greed dotted lines represent the magnitude of FRGR regularization.

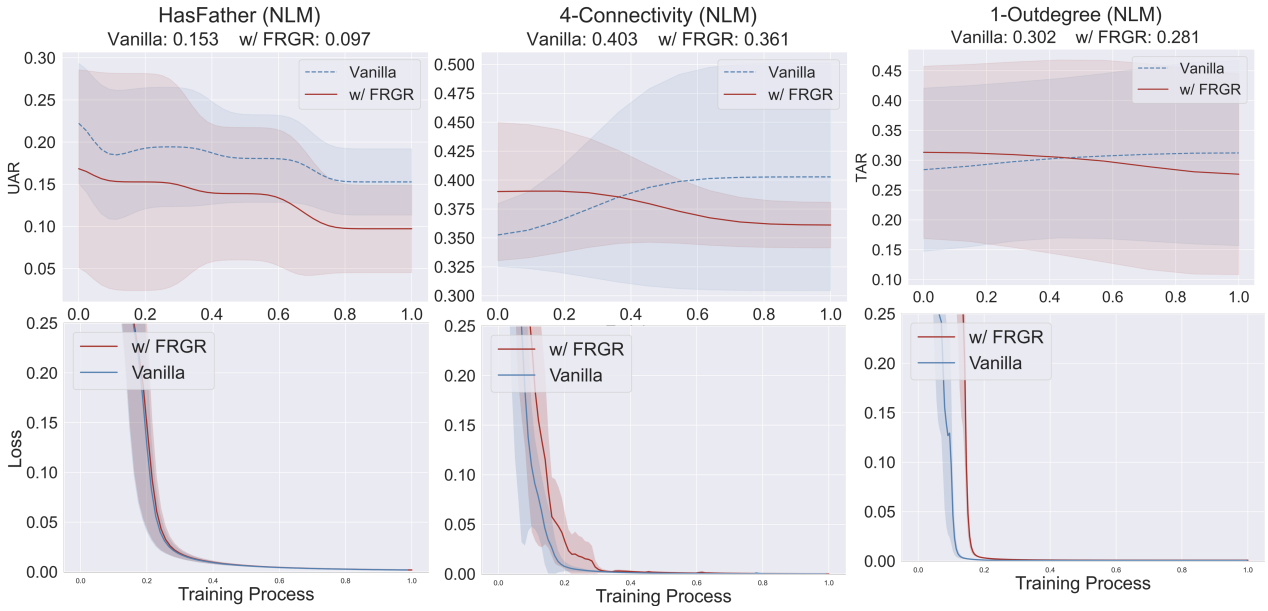

Figure 11. UAR during the training process (first row) and the training loss curve (second row) for HasFather, 4-Connectivity, and 1-Outdegree tasks. Blue lines represent training with vanilla classification loss only, and red lines represent training with FRGR regularization.