DeepProofLog: Efficient Proving in Deep Stochastic Logic Programs

Ying Jiao*[1], Rodrigo Castellano Ontiveros*[2], Luc De Raedt[1,3], Marco Gori[2], Francesco Giannini[4], Michelangelo Diligenti[2], Giuseppe Marra[1]

[1]KU Leuven, Belgium
[2] University of Siena, Italy
[3] Örebro University, Sweden
[4] Scuola Normale Superiore, Italy
firstname.lastname@kuleuven.be, firstname.lastname@unisi.it, francesco.giannini@sns.it

Abstract

Neurosymbolic (NeSy) AI aims to combine the strengths of neural architectures and symbolic reasoning to improve the accuracy, interpretability, and generalization capability of AI models. While logic inference on top of subsymbolic modules has been shown to effectively guarantee these properties, this often comes at the cost of reduced scalability, which can severely limit the usability of NeSy models. This paper introduces DeepProofLog (DPrL), a novel NeSy system based on stochastic logic programs, which addresses the scalability limitations of previous methods. DPrL parameterizes all derivation steps with neural networks, allowing efficient neural guidance over the proving system. Additionally, we establish a formal mapping between the resolution process of our deep stochastic logic programs and Markov Decision Processes, enabling the application of dynamic programming and reinforcement learning techniques for efficient inference and learning. This theoretical connection improves scalability for complex proof spaces and large knowledge bases. Our experiments on standard NeSy benchmarks and knowledge graph reasoning tasks demonstrate that DPrL outperforms existing state-of-the-art NeSy systems, advancing scalability to larger and more complex settings than previously possible.

1 Introduction

Neurosymbolic (NeSy) AI represents a promising paradigm that seeks to bridge the gap between data-driven neural architectures and structured reasoning capabilities of symbolic methods to obtain improved accuracy and enhanced interpretability. A key research direction within NeSy focuses on integrating probability with logic programming, enabling principled learning and inference under uncertainty (Manhaeve et al. 2018; Dai et al. 2019; Marra and Kuželka 2021; Pryor et al. 2022; Maene and De Raedt 2023; Yang, Ishay, and Lee 2023; De Smet et al. 2023). These systems typically perform inference in two stages: (1) apply symbolic proving to find proofs for a given query, (2) rely on probabilistic inference over these proofs to compute the probability that the query holds. This process employs the *possible world semantics* (De Raedt, Kimmig, and Toivonen 2007; Manhaeve et al. 2018), which computes the probability of a query as the sum of the probabilities of those possible worlds that have at least a proof for the query. Regrettably, this probabilistic inference task corresponds to the weighted model counting problem, which is #P-hard (Roth 1996).

To alleviate the scalability challenges in probabilistic-logic NeSy systems (Wang, Mazaitis, and Cohen 2013; Maene, Derkinderen, and De Raedt 2024; Marra et al. 2024; Jiao, De Raedt, and Marra 2024), most existing studies focus on scaling the probabilistic inference phase, assuming that derivations of queries can be computed tractably, including sampling approaches (Verreet et al. 2024) and variational approaches (Abboud, Ceylan, and Lukasiewicz 2020; Dos Martires 2021; van Krieken et al. 2023). Although these methods represent progress in the scalability of NeSy, the applicability to large-scale relational settings remains out of their reach. This limitation arises from two primary factors. First, many solutions are still ad hoc, tailored to specific tasks, and require substantial manual effort to adapt to new domains. Second, and more fundamentally, these methods do not address the computational cost of deriving the proofs.

This paper introduces DeepProofLog (DPrL), a novel system taking a different perspective to significantly advance NeSy scalability (see Figure 1). Instead of relying on possible world semantics, our work focuses on the semantics of Stochastic Logic Programs (SLPs) (Cussens 2001), which define a probability distribution over *derivations* rather than over possible worlds. The probability associated with a clause in SLPs reflects its likelihood of being used in a successful query derivation, rather than its truth in a possible world. In this perspective, SLPs do not directly model uncertainty about the world, but rather the uncertainty in the behavior of a prover operating within that world. This view offers an agent-based interpretation of inference and learning in NeSy, which, to the best of our knowledge, has not been previously explored. Furthermore, we define a formal correspondence between learning and inference in (deep) SLPs and the problem of learning optimal policies in Markov Decision Processes (MDPs) (Feinberg and Shwartz 2012). In our formulation, each derivation corresponds to a trajectory in the MDP, and each proving step is parameterized by a neural network that works as a policy by providing continuous and goal-directed guidance for selecting the next action (i.e. a rule application or variable substitution). This connection allows us to draw upon the rich body of established MDP solution methods, which have proven effective

*Equal first authorship

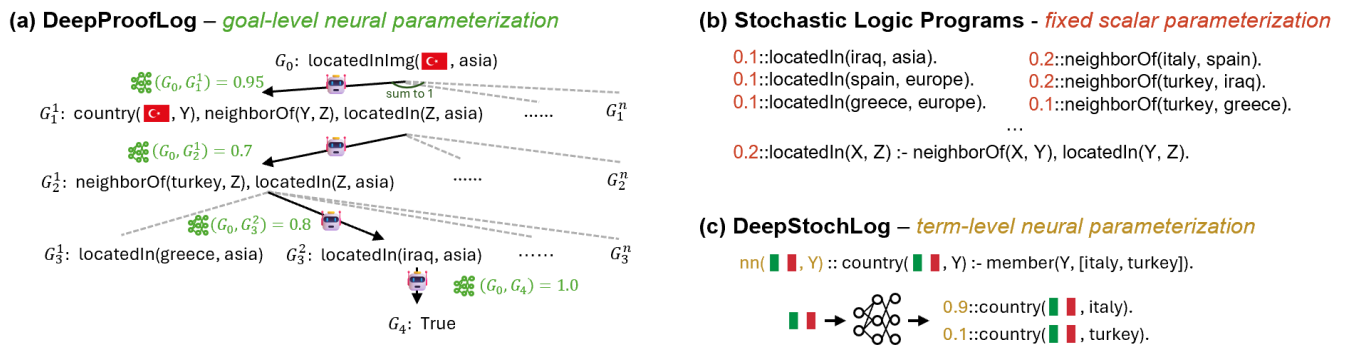

Figure 1: (a) DeepProofLog uses goal-level parameterization: a neural policy conditioned on the goal to guide proving at each resolution step. In comparison, (b) SLPs assign scalar weights to clauses for probabilistic proof search without input adaptation, and (c) DeepStochLog uses neural nets conditioned on subsymbolic terms in the query, requiring a fixed output domain.

in a variety of complex domains, such as game playing and robotics, to tackle scenarios that have traditionally been intractable for NeSy systems. In particular, when the symbolic goal space is computationally manageable, we show that dynamic programming techniques can be used for exact inference with enhanced efficiency. Conversely, when the complete goal space becomes intractable, we apply deep reinforcement learning to approximate the optimal policy through learning from sampled derivations.

To our knowledge, the only other NeSy system designed as a neural extension of SLPs is DeepStochLog (Winters et al. 2022). It is based on stochastic definite clause grammars, a type of SLPs, and introduces neural grammar rules. While DeepStochLog represents a step forward, it still faces challenges with scalability and expressiveness compared to DPrL. A more technical comparison between our method and DeepStochLog is reported in Section 7.

Contributions. This paper makes the following contributions. (i) We introduce DeepProofLog (DPrL), a novel deep SLP model that leverages neural parameterization at each proof step to provide continuous, goal-directed guidance during reasoning. (ii) We propose a theoretical mapping between derivation-based probabilistic logic reasoning and MDPs, enabling the use of powerful MDP solution techniques for efficient learning and inference in DPrL. (iii) The experiments on classic NeSy benchmarks and knowledge graph completion tasks empirically validate the effectiveness of our approach, achieving improved task performance, efficiency, and scalability with respect to other NeSy systems.

2 Preliminaries

2.1 Logic Programming

Following (Flach 1994; Lloyd 2012), we summarize the basic concepts of logic programming. A term is either a constant, a variable, or a compound term of the form $f(t_1, ..., t_n)$, where f is a functor of arity n and each t_i is a term. An expression $r(t_1, ..., t_n)$, where r is a predicate and each t_i is a term, is called an atom. For example, $\mathit{locatedIn}(X, \mathit{europe})$ is an atom where *europe* denotes a constant and *locatedIn* a binary predicate. A definite clause[1] is of the form $H \leftarrow B_1, ..., B_m$, where H is the *head* atom and $B_1, \ldots, B_m$ form the *body* atoms. A set of definite clauses $\mathcal{P}$ is called a *definite logic program*. A clause of the form $\leftarrow A_1, \ldots, A_n$ is called *goal* and represents a query to the program. A *substitution* θ such as $\{X/italy\}$ is a mapping from variables to terms. A substitution θ is a *unifier* of two atoms (same for terms) A_1 and A_2 if $A_1\theta$ (application of θ to variables of A_1) = $A_2\theta$. For instance, $\theta = \{X/italy, Y/europe\}$ unifies $A_1 = \mathit{locatedIn}(X, europe)$ and $A_2 = \mathit{locatedIn}(italy, Y)$, as $A_1\theta = A_2\theta = \mathit{locatedIn}(italy, europe)$. A substitution θ is the Most General Unifier (MGU) if any other unifier can be obtained by further instantiating θ—i.e., it makes the minimal necessary variable substitutions.

Given a logic program $\mathcal{P}$, a Selective Linear Definite (SLD) *derivation* of an initial goal G_0 is defined as a (finite or infinite) sequence of goals $G_0 \vdash G_1 \vdash G_2 ...$ where for each $i \geq 0$ we have: i) $G_i = \leftarrow A_1^i, \ldots, A_n^i$, ii) $C_i = H_i \leftarrow B_1^i, \ldots, B_m^i$ is a clause in $\mathcal{P}$, with A_1^i and H_i unifiable with MGU θ_i, iii) G_{i+1} is constructed from G_i by replacing A_1^i with the body of C_i, and by applying the substitution θ_i to each atom in the resulting goal, i.e. $G_{i+1} = (\leftarrow B_1^i, \ldots, B_m^i, A_2^i, \ldots, A_n^i)\theta_i$. We denote this resolution step by $G_{i+1} = \mathrm{res}(G_i, C_i, \theta_i)$. The process proceeds by repeatedly applying the above resolution steps until the goal reduces to the empty/True goal (successful derivation), or no further resolution is possible, thus the goal is False (failure derivation).

2.2 Stochastic SLD Resolution

Stochastic SLD resolution defines a probability distribution over derivations. This framework captures uncertainty in the resolution process and enables the learning of probabilistic models from data. As special cases, both SLPs and DeepStochLog can be derived from this approach. Their relations are illustrated in Figure 1.

Definition 1 (Probability Distribution over SLD Derivations)**.** *Let $d = (G_0, G_1, \ldots, G_n)$ be a finite SLD deriva-*

[1]We will omit "definite" as all clauses in this paper are definite.

tion. The probability of d is defined as:

$$p(d) = \prod_{i=0}^{n-1} p(G_{i+1} \mid G_i)$$

where $p(G_{i+1} \mid G_i)$ is the probability of transitioning from goal G_i to goal G_{i+1} by applying one SLD resolution step. We define $p(G_{i+1} \mid G_i)$ as the resolution transition model*, which specifies a distribution over the possible outcomes of resolving the leftmost atom in G_i.*

Given a query q, we are interested in computing its success probability under the derivation probability distribution.

Definition 2 (Success Probability of a Query)**.** *Let $\mathcal{R}(q)$ be the set of all successful derivations for a query q, i.e. derivations $d = (G_0, G_1, ..., G_n)$ with $G_0 = q$ and $G_n = True$, for some $n > 0$. The success probability of q is:*

$$p_{success}(q) = \sum_{d \in \mathcal{R}(q)} p(d) = \sum_{d \in \mathcal{R}(q)} \prod_{i=0}^{n-1} p(G_{i+1} \mid G_i) \; .$$

Stochastic Logic Programs. Pure normalized SLPs extend logic programming by associating each clause C with a fixed probability $\lambda_C \in [0, 1]$, written as $\lambda_C :: H \leftarrow B_1, \ldots, B_m$. For a predicate r, let $\mathcal{C}(r)$ denote the set of clauses with head predicate r, satisfying $\sum_{C \in \mathcal{C}(r)} \lambda_C = 1$. This defines a *scalar parameterization* of the resolution transition model: $p(G_{i+1} \mid G_i) = \lambda_{C_i}$, where C_i is the clause used in the resolution step $G_{i+1} = \text{res}(G_i, C_i, \theta_i)$.

DeepStochLog. DeepStochLog replaces fixed clause probabilities of SLPs with a neural network ρ_λ, which produces a probability distribution over possible variable substitutions. This induces a *term-level neural parameterization* of the transition model $p(G_{i+1} \mid G_i) = \rho_\lambda(X\theta_i, Y\theta_i)$, where θ_i is the MGU used in the resolution step from G_i to G_{i+1}, i.e. ρ_λ is conditioned on the instantiated input and output terms $X\theta_i, Y\theta_i$ realizing the unification step. An example is shown in Figure 1.

Limitations. SLPs and DeepStochLog both restrict the expressiveness of stochastic SLD resolution. In classical SLPs, clause probabilities λ_C are fixed per predicate and do not depend on the specific substitutions from unification, making resolution insensitive to query context or input variation. DeepStochLog extends this by conditioning clause probabilities on subsymbolic inputs (e.g., differing scores for `country(img32, Y)` vs. `country(img73, Y)`), but it requires a fixed output domain, limiting flexibility when valid substitutions depend on the query (e.g., `neighborOf(italy, Y)` vs. `neighborOf(japan, Y)`). Additionally, both methods restrict clause selection to the leftmost atom, preventing the use of global context for more informed decisions.

2.3 Markov Decision Processes

An MDP is a formal model of decision-making defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}(s)$ is a set of actions available in state s, $P(s, a, s')$ is the transition function giving the probability of transitioning from state s to s' with action a, $R(s, a, s')$ is the reward function giving the expected reward for the transition and $\gamma \in [0, 1]$ is the discount factor for future rewards. A policy $\pi(a|s)$ defines the probability of selecting action a in state s. Given a starting state s_0, solving an MDP requires finding the optimal policy π^* that maximizes the future cumulative reward, defined as the value function:

$$v(s_0) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1} \Big| S = s_0\right]$$

where the expectation is over trajectories of states and actions sampled from $P(s, a, s')$ and $\pi(a|s)$.

3 DeepProofLog

To address the limitations of previous approaches, we propose a shift in perspective: instead of parameterizing the logic program itself (e.g., via clause weights), we parameterize the resolution process. To this aim we introduce DeepProofLog, which defines clause selection probabilities conditioned on the entire current goal and enables dynamic adaptation to vary substitution spaces. DeepProofLog is a framework for differentiable, goal-conditioned SLD resolution that supports flexible and context-aware reasoning.

We define the resolution transition model in DeepProofLog as:

$$p_\lambda(G_{i+1} \mid G_i) = \frac{\exp(f_\lambda(G_i, G_{i+1}))}{\sum_{G \in \mathcal{N}(G_i)} \exp(f_\lambda(G_i, G))} \qquad (1)$$

where $\mathcal{N}(G)$ denotes the set of next goals from resolving the leftmost atom of G; $f_\lambda(G, G')$ scores the compatibility between the learned embeddings of the current goal G and candidate next goal G' (as defined in the next paragraph); and λ comprises all model parameters.

Goal Representation. To enable neural processing of symbolic goals, we construct goal embeddings hierarchically from (sub)symbolic and atomic components.

(Sub)symbolic Embeddings. Let $\mathcal{V}$ denote the set of all entities involved in reasoning, including symbolic tokens (e.g., predicates, constants, variables) and subsymbolic inputs (e.g., images). Each element $v \in \mathcal{V}$ is associated with a learnable dense embedding $\mathbf{e}_v \in \mathbb{R}^d$. The embeddings e_v for all $v \in \mathcal{V}$ are part of λ (cf. Equation 1) and trained jointly with the reasoning model. This design enables a unified embedding space in which both symbolic and subsymbolic information can interact seamlessly during inference and learning.

Atom Embeddings. For an atom $A = r(t_1, \ldots, t_n)$ with predicate r and arguments $t_1, \ldots, t_n$, we compute its embedding via a composition function: $\mathbf{e}_A = f_{\text{atom}}(\mathbf{e}_r, \mathbf{e}_{t_1}, \ldots, \mathbf{e}_{t_n})$, where f_{atom} is a learnable and differentiable function that maps the component embeddings into an atom embedding. Since the arity of a predicate is fixed, f_{atom} can be instantiated with any neural network or knowledge graph embedding model.

Goal Embeddings. The embedding of a goal $G = \leftarrow A_1, \ldots, A_n$ is computed by aggregating the embeddings of its constituent atoms: $\mathbf{e}_G = f_{\text{agg}}(\mathbf{e}_{A_1}, \ldots, \mathbf{e}_{A_n})$, where f_{agg}

is a differentiable aggregation function. Common choices include: sum, mean, or a learnable neural aggregation.

Compatibility Score. The compatibility score between goals G and G' is computed as the scalar product between their embeddings: $f_\lambda(G, G') = \mathbf{e}_G \cdot \mathbf{e}_{G'}$.

Learning. In line with similar approaches (cf. Section 2), DeepProofLog defines the query success probability like in Stochastic SLD resolution (cf. Definition 2). Moreover, our learning objective aligns with DeepStochLog,

$$\mathcal{L}(\lambda) = \sum_{(q^{(i)}, y^{(i)}) \in \mathcal{D}} \ell\left(p^\lambda_{\text{success}}(q^{(i)}), y^{(i)}\right) \tag{2}$$

where $\ell(p, y) = (1 - 2y) \cdot p$ is a linear loss prompting high success probability for positive queries ($y = 1$) and low for negative ones ($y = 0$); $p^\lambda_{\text{success}}(q^{(i)})$ denotes the success probability of query $q^{(i)}$ under the DeepProofLog parameterization; and $\mathcal{D} = \{(q^{(i)}, y^{(i)})\}_{i=1}^N$ is the labeled dataset.

Expressiveness. Before discussing learning and inference capabilities, we show in the following theorem that DeepProofLog can approximate any probability distribution over SLD derivations arbitrarily well.

Proposition 1 (Universal Approximation)**.** *Any probability distribution over SLD Derivations can be approximated with arbitrary precision by DeepProofLog.*

The proof is in Appendix A.1.

4 Markov Decision Processes for DeepProofLog

DeepProofLog's novel perspective of parameterizing stochastic SLD resolution opens the door to interpreting learning and reasoning in any (deep) SLP as learning an optimal policy within a logic program-based environment of an MDP. In the following we show how DeepProofLog instantiates an MDP by defining the components $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$.

SLD-resolution as MDP. We formalize the SLD-resolution as an MDP, where each component is defined based on a dataset of labeled queries $\mathcal{D} = \{(q^{(i)}, y^{(i)})\}_{i=1}^N$, with $y^{(i)} \in \{1, 0\}$ indicating whether $q^{(i)}$ is a positive or negative query.

States. The state space $\mathcal{S}$ is defined by $\mathcal{S} = \mathcal{Q} \cup \mathcal{G} \cup \{\textit{True}, \textit{False}\}$, where $\mathcal{Q} = \{q^{(i)}\}_{i=1}^N$ is the set of initial queries in $\mathcal{D}$, and $\mathcal{G}$ is the set of all intermediate sub-goals that can arise from any derivation starting from any $q^{(i)}$.

Actions. Let $G = \leftarrow A_1, \ldots, A_n$ be a state, and let $\mathcal{P}$ be the definite logic program. The action space $\mathcal{A}(G)$ is defined as $\mathcal{A}(G) = \{(C, \theta) |\ C \in \mathcal{P}, \theta = \text{MGU}(A_1, \text{head}(C))\}$. Additionally, we introduce the special *False* action to enable the model to learn early abandonment. More details on the advantages and implementation are provided in Appendix B.2.

Transition Function. The transition function is deterministic and follows from logic resolution: given $G_{i+1} = \text{res}(G_i, C, \theta)$, the transition probability is defined as $P(G' \mid G_i, C, \theta) = \mathbf{1}\{G' = G_{i+1}\}$.

Reward. To define the reward, we introduce an auxiliary set of labeled states, where each state is paired with the label of the original query it stems from, i.e., $\tilde{\mathcal{S}} = \{(G, y^{(i)}) \mid G \in \mathcal{S}, y^{(i)} \in \{0, 1\}$ is the label of $q^{(i)} \in \mathcal{Q}\}$. Note that the labeled states $\tilde{\mathcal{S}}$ are only used for reward computation, and they do not affect the state space $\mathcal{S}$, and are not accessed by the policy. The reward function $R : \tilde{\mathcal{S}} \to \mathbb{R}$ is defined as

$$R(G, y^{(i)}) = \begin{cases} 2y^{(i)} - 1 & \text{if } G = \textit{True} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Hence, proving a positive query yields a reward of $+1$, while a negative query yields -1. No immediate reward is given for intermediate states.

Policy and Value Function. In our MDP formulation, the policy π_λ defines a probability distribution over $\mathcal{A}(G_i)$ for a goal G_i. Each action $a = (C, \theta) \in \mathcal{A}(G_i)$ to G_i deterministically yields the next goal $G_{i+1} = \text{res}(G_i, a)$, establishing a one-to-one mapping between actions and successor states. Formally, this yields:

$$\pi_\lambda(a \mid G_i) = p_\lambda(G_{i+1} \mid G_i)$$

This equation highlights that a policy guiding the SLD resolution process is parameterized exactly by the transition probabilities of our resolution model (cf. Equation 1). This correspondence arises from the semantics of SLPs: clause probabilities represent the likelihood that a clause is selected by a prover - here modeled as an agent governed by π_λ - when proving a query.

As in standard MDPs, the value function of a policy is defined as its expected cumulative reward. While value functions generally follow the Bellman recursion, our SLD resolution MDP allows a simplified formulation for the value of a non-terminal goal G with label y (see Appendix A.2 for derivation):

$$V^{\pi_\lambda}(G, y) = \sum_{a \in \mathcal{A}(G)} \pi_\lambda(\text{res}(G, a), G) \cdot \cdot V^{\pi_\lambda}(\text{res}(G, a), y)\ , \tag{4}$$

with boundary condition $V^{\pi_\lambda}(G, y) = R(G, y)$ for a terminal goal G.

Example. We illustrate the introduced terms with the following example. Consider a logic program $\mathcal{P} = \{C_1, C_2, C_3, C_4, C_5\}$, where the clauses are defined as: $C_1 = \textit{locIn}(X, Y) \leftarrow \textit{neighOf}(X, Z), \textit{locIn}(Z, Y)$, $C_2 = \leftarrow \textit{neighOf}(\textit{it}, \textit{fr})$, $C_3 = \leftarrow \textit{locIn}(\textit{fr}, \textit{eu})$, $C_4 = \leftarrow \textit{locIn}(\textit{tr}, \textit{gr})$, and $C_5 = \leftarrow \textit{locIn}(\textit{gr}, \textit{eu})$. Consider a positive query $q^{(0)} = G_0^{(0)} = \leftarrow \textit{locIn}(\textit{it}, \textit{eu})$ with label $y^{(0)} = 1$. The corresponding action space is $\mathcal{A}(G_0^{(0)}) = \{(C_1, \{X/\textit{it}, Z/\textit{eu}\})\}$. Applying the action gives the next goal $G_1^{(0)} = \leftarrow \textit{neighOf}(\textit{it}, Y), \textit{locIn}(Y, \textit{eu})$, with intermediate reward $R(G_1^{(0)}) = 0$. Assuming the episode ends after n resolution steps with $G_n^{(0)} = \textit{True}$, the final reward is $R(G_n^{(0)}, y^{(0)}) = 2 \cdot 1 - 1 = 1$. Now consider a negative query $q^{(1)} = G_0^{(1)} = \leftarrow \textit{locIn}(\textit{tr}, \textit{eu})$ with label $y^{(1)} = 0$. If the episode ends in $G_m^{(1)} = \textit{True}$ after m resolution steps, the final reward is $R(G_m^{(1)}, y^{(1)}) = 2 \cdot 0 - 1 = -1$.

5 Learning in DeepProofLog

Learning in DeepProofLog is equivalent to learning the optimal policy of the corresponding MDP defined in Section 4. Let the objective be $J(\pi_\lambda) = \sum_{i=1}^{N} [V^{\pi_\lambda}(q^{(i)}, y^{(i)})]$ (Sutton and Barto 1999). Given our reward design in Equation 3 and setting the discount factor $\gamma = 1$, we show in Appendix A.3 that the objective function simplifies to:

$$J(\pi_\lambda) = \sum_{i=1}^{N} (2y^{(i)} - 1) \cdot p^{\pi_\lambda}_{\text{success}}(q^{(i)})$$

where $p^{\pi_\lambda}_{\text{success}}(q^{(i)})$ is the success probability of $q^{(i)}$ under π_λ.

Proposition 2. *Maximizing the expected return $J(\pi_\lambda)$ in our MDP formulation corresponds to minimizing the objective $\mathcal{L}(\lambda)$ defined in Equation 2.*

Maximizing $J(\pi_\lambda)$ also aligns with minimizing the cross-entropy loss commonly used in NeSy learning; see Appendix A.3 for the proof.

Exact Inference via Dynamic Programming. As demonstrated in our experiments, standard NeSy benchmarks, often perceived to involve combinatorial search spaces, can be encoded to yield a manageable state space. In such cases, dynamic programming enables significantly better scalability without introducing approximation. By reasoning in terms of the state-space size of the corresponding MDP, our formulation helps to design nested symbolic components that keep the size tractable and support efficient exact inference.

Approximate Inference via Policy Gradient. When the full goal space is intractable, the mapping between SLP derivations and MDPs enables the application of reinforcement learning techniques to learn the clause selection policy. Prominent algorithms such as REINFORCE (Williams 1992), Actor-Critic (Sutton et al. 2000), and Proximal Policy Optimization (PPO) (Schulman et al. 2017) can be adapted. For example, we employ PPO to optimize the neural policy, using the standard clipped surrogate objective. Alongside the policy, we parameterize a neural value function with the same architecture to estimate returns and reduce variance during training. While value-based methods are also applicable in principle, we leave their exploration to future work.

6 Experiments

We present two sets of experiments. First, we assess our approach on a neurosymbolic task that learns the perception given the background knowledge, showing that DeepProofLog (DPrL) outperforms advanced approximate inference systems in both accuracy and scalability, while scaling better than exact inference systems like DeepStochLog. The second set of experiments addresses explainable knowledge graph (KG) completion. Here, DPrL learns from data to generate high-probability proofs for positive queries and low-probability proofs for negative ones. The fact that each classification is expressed by a proof makes DPrL decisions fully interpretable. Comparative experiments against state-of-the-art proof-based KG systems show that our method scales to large KGs without compromising its predictive performance. All code and data will be publicly available upon publication.

6.1 MNIST Addition

MNIST addition is a popular NeSy benchmark, where the model predicts the sum of two numbers represented as sequences of N MNIST digit images. The only supervision provided is the final sum, without explicit labels for individual digits. This task serves as a testbed for evaluating the scalability of DPrL, as the reasoning complexity increases with the number of digits N.

We explore both dynamic programming (DP) and policy gradient (PG) approaches for this task. For DP, we compute the exact optimal policy by maximizing the value function of the MDP induced by the logic program, following the formulation in Equation 4. For PG, we implement an off-policy REINFORCE algorithm where the policy is a neural digit classifier. The logic program structure naturally constrains the set of valid actions at each step, reducing sampling variance. The logic programs are provided in Appendix C.1.

We compare with state-of-the-art NeSy methods along two dimensions: predicted sum accuracy and training time. The baselines include three exact systems: KLay (Maene, Derkinderen, and Martires 2025), DeepStochLog, and DeepSoftLog (Maene and De Raedt 2023); and three approximate systems: Scallop (Huang et al. 2021), A-NeSI (van Krieken et al. 2023), and EXAL (Verreet et al. 2024). The results are summarized in Table 1, with details on the experimental setups and hyperparameters provided in Appendix C.1.

The DP variant achieves accuracy comparable to existing exact NeSy systems, while significantly outperforming both exact and approximate baselines in terms of training efficiency. We observed that no other previous method successfully scales to $N = 100$ without timing out. In the approximate setting, our PG variant achieves the highest accuracy. It scales beyond $N = 100$, demonstrating strong computational efficiency, but converges to a suboptimal policy due to the vast combinatorial search space.

6.2 Explainable Knowledge Graph Completion

We evaluate our approach on popular benchmarks for KG completion: Family (Kok and Domingos 2007) and WN18RR (Dettmers et al. 2018). The Family dataset focuses on modeling familial relationships, and the WN18RR dataset is a subset of WordNet (WN18) created to provide a more challenging benchmark by removing inverse relations. More information about the dataset statistics can be found in the Appendix C.2. All methods have been tested with 200 randomly sampled negative corruptions per query.

Table 2 compares DPrL against both KGE baselines (ComplEx (Trouillon et al. 2016) and RotatE (Sun et al. 2019)) and other logic-based NeSy systems like Semantic-based Regularization (SBR) (Diligenti, Gori, and Sacca 2017), R2N (Marra, Diligenti, and Giannini 2025) and DCR (Barbiero et al. 2023). Please note that these NeSy systems would not scale to the considered datasets without relying on custom approximate grounding techniques (Ontiveros et al.

	N		4	15	100	200	500
	Reference		92.9	75.8	15.4	2.4	0.009
NeSy Exact	KLay	Time	T/O				
	DeepStochLog	Acc.	92.7 ± 0.6	T/O			
		Time	141.2 ± 30.8				
	DeepSoftLog	Acc.	93.5 ± 0.6	77.1 ± 1.6	T/O		
		Time	879.9 ± 45.3	1324.9 ± 72.6			
	DPrL (DP)	Acc.	94.0 ± 0.3	80.8 ± 1.1	37.8 ± 2.9	23.2 ± 3.0	6.0 ± 4.9
		Time	25.4 ± 6.3	41.8 ± 3.7	105.8 ± 21.8	128.0 ± 33.3	469.0 ± 51.6
NeSy Approximate	Scallop	Acc.	92.3 ± 0.7	T/O			
		Time	73.6 ± 3.9				
	A-NeSI	Acc.	92.6 ± 0.8	75.9 ± 2.2	T/O		
		Time	70.1 ± 3.1	916.5 ± 35.8			
	EXAL	Acc.	91.8 ± 0.8	73.3 ± 2.1	T/O		
		Time	32.0 ± 1.7	145.1 ± 6.8			
	DPrL (PG)	Acc.	93.5 ± 0.4	76.9 ± 1.9	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
		Time	25.7 ± 8.6	49.7 ± 12.0	282.6 ± 20.5	295.8 ± 46.0	865.0 ± 67.9

Table 1: Test accuracy and training times (in minutes, shown in parentheses) on MNIST addition with sequence length N. A time-out (T/O) of 24 hours (1440 minutes) is applied. Reference indicates the accuracy on the sum obtainable by a single-digit classifier with accuracy of 0.9907, e.g., Reference=0.9907^{2N} (Maene and De Raedt 2023).

		Family				WN18RR			
		MRR	Hits@1	Hits@3	Hits@10	MRR	Hits@1	Hits@3	Hits@10
KGE	ComplEx	0.964	0.938	0.991	0.994	0.479	0.438	0.478	0.540
	RotatE	0.968	0.943	0.994	0.995	0.833	0.787	0.863	0.913
NeSy	R2N_{SG}	0.985	0.981	0.989	0.989	0.724	0.699	0.733	0.755
	DCR_{SG}	0.975	0.956	0.994	0.995	0.817	0.754	0.862	0.922
	SBR_{SG}	0.981	0.966	0.995	0.995	0.840	0.784	0.879	0.935
	DPrL (PG)	0.986	0.979	0.994	0.995	0.834	0.784	0.868	0.918

Table 2: Comparison of KGE and NeSy Methods on Family and WN18RR Datasets.

2025), which make it possible to perform approximate reasoning using these methodologies. We indicate these variants as *R2N*$_{\text{SG}}$, *DCR*$_{\text{SG}}$, and *SBR*$_{\text{SG}}$. However, these fast grounding techniques require a manual fine-tuning of their parameters, whereas DPrL automatically learns the best exploration strategy for each dataset.

DPrL uses Proximal Policy Optimization (PPO) (Schulman et al. 2017) to navigate the complex, large-scale action spaces emerging in KG reasoning. The model computes the scores $p^{\lambda}_{\text{success}}(q)$ for each query q so that the standard evaluation metrics for each task (MRR, Hits@N) can be computed (see Appendix C.2 for the metric definitions). All NeSy methods, including DPrL, use RotatE output as a prior, learning a logic-based adjustment on top of it.

For the Family dataset, all NeSy systems improve performance over the baseline. DPrL is particularly effective, especially on MRR and Hits@1 metrics. A similar trend is observed in the WN18RR dataset, where DPrL and SBR outperformed other methods.

Unlike the partially latent reasoning done by R2N or DCR, DPrL is fully interpretable as each classification decision is the result of the execution of a proof, which can be inspected by a human. Table 3 showcases two examples of explanations in the form of proofs discovered by DPrL: a depth-3 proof for the query synset_domain_topic_of(09788237, 08441203) from the WN18RR dataset, and a proof for the query uncle(2266, 2252) in the Family dataset.

Methods based on a complete or approximate grounding technique, like the ones proposed by (Ontiveros et al. 2025), would need to instantiate and process a combinatorially large number of ground formulas to find a proof at high depth. In contrast, DPrL can avoid an exhaustive search by learning an efficient proof-finding policy, which efficiently navigates deep reasoning paths. For example, on the Family dataset, the smart grounding techniques proposed by (Ontiveros et al. 2025) instantiate 18071, 45466, 181095 groundings for the test queries, when allowing proofs at depth 1,2,3, respectively. On the other hand, DPrL needs to evaluate only 12334 ground formulas to perform full inference over the test set.

In conclusion, DPrL demonstrates superior scalability over other NeSy models capable of learning perception and structure, such as DeepSoftLog (which fails to scale to

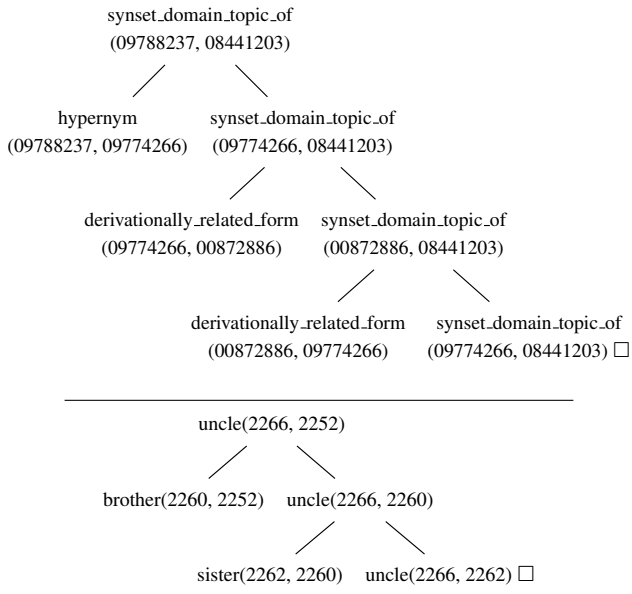

Table 3: Examples of multi-step proof trees discovered by DeepProofLog for the Family and WN18RR datasets.

datasets presented here), while preserving the explainability and interpretability advantages of logic-based NeSy approaches.

7 Related Work

DeepStochLog. DeepStochLog extends SLPs by using neural networks to parameterize clause probabilities under derivation-based semantics. However, its exact inference leads to the combinatorial explosion of proof paths, making it intractable for complex tasks. It also assumes fixed output domains and conditions neural predictions on partial goals, limiting flexibility and context awareness. In contrast, DeepProofLog factorizes the derivation distribution over resolution steps, conditions each decision on the full current goal, and supports variable output domains. This results in better scalability (linear in derivation steps and matching unification), more informed decisions, and greater expressivity.

Scaling inference in NeSy. Efforts to improve the scalability of exact NeSy systems include DeepStochLog, which adopts a more efficient derivation-based semantics, and KLay (Maene, Derkinderen, and Martires 2025), which introduces GPU-parallelizable arithmetic circuits. Approximate NeSy methods include search-based techniques (Manhaeve, Marra, and De Raedt 2021; Huang et al. 2021) and random walks with restart (Wang, Mazaitis, and Cohen 2013), which limit inference to parts of the proof space but risk bias. Sampling-based approaches (Verreet, De Smet, and Sansone 2025) avoid full model counting but are still limited by grounding. Variational inference (Abboud, Ceylan, and Lukasiewicz 2020; Dos Martires 2021; van Krieken et al. 2023) replaces logic inference with learned predictors, trading interpretability for speed. Parametrized grounding (Castellano Ontiveros et al. 2025) offers a tunable balance between scalability and expressivity but lacks general guarantees like DeepProofLog.

Automated Theorem Proving. Automated Theorem Proving (ATP) aims to find a proof for a given query within deterministic logical frameworks. Recent advances in ATP integrate learning-based heuristics to guide proof search (Rocktäschel and Riedel 2017; Kaliszyk et al. 2018; Rawson and Reger 2019; Zombori, Urban, and Brown 2020; Lample et al. 2022). Unlike ATP systems, DeepProofLog operates in a probabilistic setting and learns from labeled data. Rather than determining provability, it optimizes inference by maximizing the probability of positive queries and minimizing that of negatives.

Reinforcement Learning in KG Completion. Reinforcement learning (RL) has been applied to knowledge graph (KG) completion mainly via path-finding models like DeepPath (Xiong, Hoang, and Wang 2017) and MINERVA (Das et al. 2017), which train agents to navigate KGs by sequentially selecting entities and relations. These approaches map relation-path discovery to an MDP but are specific to KG structures. While paths can serve as explanations, they may be complex and hard to interpret. In contrast, DeepProofLog maps SLPs to MDPs, enabling RL-guided proof search across any domain expressible with SLPs, offering greater generality beyond KGs.

8 Conclusions

We introduce DeepProofLog, a scalable and flexible NeSy framework that bridges the gap between expressive logical representations and efficient, data-driven learning. DeepProofLog leverages SLPs with derivation-based semantics and introduces a more expressive neural parameterization by conditioning on the full goal at each resolution step. A primary contribution is the novel and formal correspondence between inference and learning in our deep SLPs and policy optimization in MDPs, enabling the use of dynamic programming and reinforcement learning techniques to tackle previously intractable NeSy problems.

The presented method shares a limitation with prior work: clause selection is restricted to the leftmost atom. While this preserves expressivity and completeness of the proofs, we plan to explore a more flexible selection, which could more efficiently prune failing derivations. Our current implementation uses standard policy gradients; future work will consider advanced RL techniques and broader evaluation settings. We also see potential in extending to possible world–based NeSy methods like Markov Logic Networks (Richardson and Domingos 2006) and in studying automatic logic program reformulations to reduce the search space (Leuschel and Bruynooghe 2002).

9 Acknowledgments

This work has been supported by the EU Framework Program for Research and Innovation Horizon under the Grant Agreement No 101073307 (MSCA-DN LeMuR). F. Giannini acknowledges support from the Partnership Extended

PE00000013 - FAIR - Future Artificial Intelligence Research - Spoke 1 Human-centered AI and ERC-2018-ADG G.A. 834756 XAI: Science and technology for the eXplanation of AI decision making. L. De Raedt is also supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

Y. Jiao wants to thank Gabriele Venturato and Lennert De Smet for stimulating discussions.

References

Abboud, R.; Ceylan, I.; and Lukasiewicz, T. 2020. Learning to reason: Leveraging neural networks for approximate DNF counting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 04, 3097–3104.

Barbiero, P.; Ciravegna, G.; Giannini, F.; Zarlenga, M. E.; Magister, L. C.; Tonda, A.; Lió, P.; Precioso, F.; Jamnik, M.; and Marra, G. 2023. Interpretable neural-symbolic concept reasoning. In *International Conference on Machine Learning*, 1801–1825. PMLR.

Castellano Ontiveros, R.; Giannini, F.; Gori, M.; Marra, G.; and Diligenti, M. 2025. Grounding Methods for Neural-Symbolic AI. *arXiv e-prints*, arXiv–2507.

Cussens, J. 2001. Parameter estimation in stochastic logic programs. *Machine Learning*, 44: 245–271.

Dai, W.-Z.; Xu, Q.; Yu, Y.; and Zhou, Z.-H. 2019. Bridging machine learning and logical reasoning by abductive learning. *Advances in Neural Information Processing Systems*, 32.

Das, R.; Dhuliawala, S.; Zaheer, M.; Vilnis, L.; Durugkar, I.; Krishnamurthy, A.; Smola, A.; and McCallum, A. 2017. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. *arXiv preprint arXiv:1711.05851*.

De Raedt, L.; Kimmig, A.; and Toivonen, H. 2007. ProbLog: A probabilistic Prolog and its application in link discovery. In *Proceedings of the 20th international joint conference on artificial intelligence (IJCAI)*, 2462–2467.

De Smet, L.; Dos Martires, P. Z.; Manhaeve, R.; Marra, G.; Kimmig, A.; and De Readt, L. 2023. Neural probabilistic logic programming in discrete-continuous domains. In *Uncertainty in Artificial Intelligence*, 529–538. PMLR.

Dettmers, T.; Minervini, P.; Stenetorp, P.; and Riedel, S. 2018. Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI conference*.

Diligenti, M.; Gori, M.; and Sacca, C. 2017. Semantic-based regularization for learning and inference. *Artificial Intelligence*, 244: 143–165.

Dos Martires, P. Z. 2021. Neural Semirings. In *NeSy*, 94–103.

Feinberg, E. A.; and Shwartz, A. 2012. *Handbook of Markov decision processes: methods and applications*, volume 40. Springer Science & Business Media.

Flach, P. 1994. *Simply logical: intelligent reasoning by example*. John Wiley & Sons, Inc.

Huang, J.; Li, Z.; Chen, B.; Samel, K.; Naik, M.; Song, L.; and Si, X. 2021. Scallop: From probabilistic deductive databases to scalable differentiable reasoning. *Advances in Neural Information Processing Systems*, 34: 25134–25145.

Jiao, Y.; De Raedt, L.; and Marra, G. 2024. Valid Text-to-SQL Generation with Unification-Based DeepStochLog. In *International Conference on Neural-Symbolic Learning and Reasoning*, 312–330. Springer.

Kaliszyk, C.; Urban, J.; Michalewski, H.; and Olšák, M. 2018. Reinforcement learning of theorem proving. *Advances in Neural Information Processing Systems*, 31.

Kok, S.; and Domingos, P. 2007. Statistical predicate invention. In *ICML*.

Lample, G.; Lacroix, T.; Lachaux, M.-A.; Rodriguez, A.; Hayat, A.; Lavril, T.; Ebner, G.; and Martinet, X. 2022. Hypertree proof search for neural theorem proving. *Advances in neural information processing systems*, 35: 26337–26349.

Leuschel, M.; and Bruynooghe, M. 2002. Logic program specialisation through partial deduction: Control issues. *Theory and Practice of Logic Programming*, 2(4-5): 461–515.

Lloyd, J. W. 2012. *Foundations of logic programming*. Springer Science & Business Media.

Maene, J.; and De Raedt, L. 2023. Soft-unification in deep probabilistic logic. *Advances in Neural Information Processing Systems*, 36: 60804–60820.

Maene, J.; Derkinderen, V.; and De Raedt, L. 2024. On the hardness of probabilistic neurosymbolic learning. *arXiv preprint arXiv:2406.04472*.

Maene, J.; Derkinderen, V.; and Martires, P. Z. D. 2025. KLay: Accelerating Arithmetic Circuits for Neurosymbolic AI. In *ICLR*. OpenReview.net.

Manhaeve, R.; Dumancic, S.; Kimmig, A.; Demeester, T.; and De Raedt, L. 2018. Deepproblog: Neural probabilistic logic programming. *Advances in neural information processing systems*, 31.

Manhaeve, R.; Marra, G.; and De Raedt, L. 2021. Approximate inference for neural probabilistic logic programming. In *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning*, 475–486. IJCAI Organization.

Marra, G.; Diligenti, M.; and Giannini, F. 2025. Relational Reasoning Networks. *Knowledge-Based Systems*, 310: 112822.

Marra, G.; Dumančić, S.; Manhaeve, R.; and De Raedt, L. 2024. From statistical relational to neurosymbolic artificial intelligence: A survey. *Artificial Intelligence*, 104062.

Marra, G.; and Kuželka, O. 2021. Neural markov logic networks. In *Uncertainty in Artificial Intelligence*, 908–917. PMLR.

Ontiveros, R. C.; Giannini, F.; Gori, M.; Marra, G.; and Diligenti, M. 2025. Grounding Methods for Neural-Symbolic AI. arXiv:2507.08216.

Pryor, C.; Dickens, C.; Augustine, E.; Albalak, A.; Wang, W.; and Getoor, L. 2022. Neupsl: Neural probabilistic soft logic. *arXiv preprint arXiv:2205.14268*.

Rawson, M.; and Reger, G. 2019. A neurally-guided, parallel theorem prover. In *Frontiers of Combining Systems: 12th International Symposium, FroCoS 2019, London, UK, September 4-6, 2019, Proceedings 12*, 40–56. Springer.

Richardson, M.; and Domingos, P. 2006. Markov logic networks. *Machine learning*, 62(1): 107–136.

Rocktäschel, T.; and Riedel, S. 2017. End-to-end differentiable proving. *Advances in neural information processing systems*, 30.

Roth, D. 1996. On the hardness of approximate reasoning. *Artificial intelligence*, 82(1-2): 273–302.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sun, Z.; Deng, Z.-H.; Nie, J.-Y.; and Tang, J. 2019. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197*.

Sutton, R. S.; and Barto, A. G. 1999. Reinforcement learning: An introduction. *Robotica*, 17(2): 229–235.

Sutton, R. S.; McAllester, D.; Singh, S.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, volume 13.

Trouillon, T.; Welbl, J.; Riedel, S.; Gaussier, É.; and Bouchard, G. 2016. Complex embeddings for simple link prediction. In *International conference on machine learning*, 2071–2080. PMLR.

van Krieken, E.; Thanapalasingam, T.; Tomczak, J.; Van Harmelen, F.; and Ten Teije, A. 2023. A-NeSi: A scalable approximate method for probabilistic neurosymbolic inference. *Advances in Neural Information Processing Systems*, 36: 24586–24609.

Verreet, V.; De Smet, L.; De Raedt, L.; and Sansone, E. 2024. EXPLAIN, AGREE, LEARN: Scaling Learning for Neural Probabilistic Logic. *ECAI 2024*, 1349–1356.

Verreet, V.; De Smet, L.; and Sansone, E. 2025. EXPLAIN, AGREE and LEARN: A Recipe for Scalable Neural-Symbolic Learning. In *ICML Workshop onKnowledge and Logical Reasoning in the Era of Data-driven Learning*.

Wang, W. Y.; Mazaitis, K.; and Cohen, W. W. 2013. Programming with personalized pagerank: a locally groundable first-order probabilistic logic. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2129–2138.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3): 229–256.

Winters, T.; Marra, G.; Manhaeve, R.; and De Raedt, L. 2022. Deepstochlog: Neural stochastic logic programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, 10090–10100.

Xiong, W.; Hoang, T.; and Wang, W. Y. 2017. Deeppath: A reinforcement learning method for knowledge graph reasoning. *arXiv preprint arXiv:1707.06690*.

Yang, Z.; Ishay, A.; and Lee, J. 2023. Neurasp: Embracing neural networks into answer set programming. *arXiv preprint arXiv:2307.07700*.

Zombori, Z.; Urban, J.; and Brown, C. E. 2020. Prolog Technology Reinforcement Learning Prover: (System Description). In *International Joint Conference on Automated Reasoning*, 489–507. Springer.

A Proofs

A.1 Proof of Proposition 1

Definition 3 (Derivation Tree)**.** *Let $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ be a tree of derivations rooted at $G_0 \in \mathcal{V}$, where each node $G \in \mathcal{V}$ is a goal with a unique representation $\mathbf{e}_G$. We assume all nodes are unique (no goal appears twice in the derivation).*

Definition 4 (Target and network distributions)**.** *Let $p(G'|G)$ be the distribution computed by the network for $\forall (G, G') \in \mathcal{E}$, and $p^t(G'|G) \in (0, 1]$ be the target distribution we want to approximate.*

We aim to show that for any valid target probability distribution p^t and any $\epsilon > 0$, there exist network parameters such that:

$$|p(G'|G) - p^t(G'|G)| < \epsilon \quad \forall (G, G') \in \mathcal{E}.$$

Proof. Given the definition of $p(G'|G)$ in Section 3, we need to find scalar products $\mathbf{e}_G \cdot \mathbf{e}_{G'}$ such that:

$$p(G'|G) = \frac{\exp(\mathbf{e}_{G'} \cdot \mathbf{e}_G)}{\sum_{G_c \in \mathcal{A}(G)} \exp(\mathbf{e}_{G_c} \cdot \mathbf{e}_G)} \approx p^t(G'|G) \, .$$

If $\exp(\mathbf{e}_{G'} \cdot \mathbf{e}_G) \approx p^t(G'|G)$, $\forall (G, G') \in \mathcal{E}$, then the model is a good approximation of the target:

$$\begin{aligned} p(G'|G) &= \frac{\exp(\mathbf{e}_{G'} \cdot \mathbf{e}_G)}{\sum_{G_c \in \mathcal{A}(G)} \exp(\mathbf{e}_{G_c} \cdot \mathbf{e}_G)} \\ &\approx \frac{\exp(\mathbf{e}_{G'} \cdot \mathbf{e}_G)}{\sum_{G_c \in \mathcal{A}(G)} p^t(G_c|G)} = \\ &= \exp(\mathbf{e}_{G'} \cdot \mathbf{e}_G) \approx p^t(G'|G) \, . \end{aligned}$$

Therefore, $\mathbf{e}_{G'} \cdot \mathbf{e}_G = \log p^t(G'|G)$ are sufficient conditions to find a perfect approximation for the target distribution.

We now show that it is possible to construct the embeddings $\mathbf{e}_G$ so that they respect such constraints. Let d be an embedding size such that $b_G \in \mathbb{R}^d$ is an orthonormal basis for the goals G. This basis exists by assuming that the embedding space is at least as large as the number of nodes (distinct derivation steps) in the tree, e.g., $d \geq |\mathcal{V}|$.

If this holds, the goal representations can be recursively constructed by first defining the embedding for the root node G_0 to be its own basis vector: $\mathbf{e}_{G_0} = b_{G_0}$. Then, we define the embeddings to be recursively computed as: $\mathbf{e}_{G'} = \log p^t(G'|G) \cdot b_G + b_{G'}$. Since the nodes G, G' are distinct (and so is also G's parent), their corresponding basis vectors are orthogonal, and it can be shown by basic math that:

$$\mathbf{e}_{G'} \cdot \mathbf{e}_G = \log p^t(G'|G)$$

Therefore, a valid geometric configuration of target vectors $\mathbf{e}_G$ exists. By the Universal Approximation Theorem for Neural Networks, and since node labels are not repeated (all goals are distinct in the derivation tree), a sufficiently powerful MLP can learn the finite mapping from each unique input identifier G to the corresponding target embeddings $\mathbf{e}_G$ with arbitrary precision. □

A.2 Derivation of Recursive Value Function

Following the standard MDP formulation,

$$V^{\pi_\lambda}(G, y) = \sum_{a \in \mathcal{A}(G)} \pi_\lambda(\text{res}(G, a), G) \sum_{G'} P(G' \mid G, a) \cdot [R(G, y) + \gamma V^{\pi_\lambda}(G', y)] \, ,$$

we simplify this in our SLD-resolution MDP (cf. Section 4), where transitions are deterministic and zero rewards for non-terminal states. Setting $\gamma = 1$, the value function for non-terminal G simplifies to

$$V^{\pi_\lambda}(G, y) = \sum_{a \in \mathcal{A}(G)} \pi_\lambda(\text{res}(G, a), G) \cdot V^{\pi_\lambda}(\text{res}(G, a), y) \, ,$$

with terminal condition $V^{\pi_\lambda}(G, y) = R(G, y)$.

A.3 Alignment between MDP Objective and Cross-entropy Loss

In NeSy systems, learning is often supervised via cross-entropy between predicted success probabilities and binary labels (Winters et al. 2022). We now show that, under our MDP formulation in Section 4, the objective that maximizes the expected cumulative reward is equivalent to minimizing cross-entropy loss on a labeled dataset $\mathcal{D} = \{(q^{(i)}, y^{(i)})\}_{i=1}^N$.

NeSy objective. The NeSy systems minimize the expected cross-entropy loss:

$$\mathcal{L}(\theta) = -\sum_{i=1}^{N} \Big[\, y^{(i)} \log p^\lambda_{\text{success}}(q^{(i)}) + (1 - y^{(i)}) \log \Big(1 - p^\lambda_{\text{success}}(q^{(i)})\Big) \Big].$$

MDP objective. Given the recursive value function from Appendix A.2, the expected return of a query q with label y can be unrolled as:

$$\begin{aligned} V^{\pi_\lambda}(q, y) &= \sum_{a_0, \ldots, a_{T-1}} \prod_{t=0}^{T-1} \pi_\lambda(G_{t+1}, G_t) \cdot V^{\pi_\lambda}(G_T, y), \\ &= \sum_{a_0, \ldots, a_{T-1}} \prod_{t=0}^{T-1} \pi_\lambda(G_{t+1}, G_t) \cdot R(G_T, y), \end{aligned}$$

where $G_{t+1} = \text{res}(G_t, a_t)$, and we assume the derivation proceeds for T steps, reaching terminal goal G_T at depth T.

From the reward design in Equation 3, only successful derivations that terminate in *True* yield non-zero rewards. By the definition of query success probability in Definition 2, the value simplifies to:

$$\begin{aligned} V^{\pi_\lambda}(q, y) &= \sum_{a_0, \ldots, a_{T-1}} \prod_{t=0}^{T-1} \pi_\lambda(G_{t+1}, G_t) \cdot R(\textit{True}, y), \\ &= R(\textit{True}, y) \cdot \sum_{d \in \mathcal{R}(q)} p^\lambda(d) = (2y - 1) p^\lambda_{\text{success}}(q). \end{aligned}$$

Thus, the MDP objective becomes:

$$J(\lambda) = \sum_{i=1}^{N} V^{\pi_\lambda}(q^{(i)}, y^{(i)}) = \sum_{i=1}^{N} (2y^{(i)} - 1) p^\lambda_{\text{success}}(q^{(i)}).$$

Proof. We now compare the gradients of the two objectives. Let $P_i := p^{\lambda}_{\text{success}}(q^{(i)})$, and apply the chain rule:

$$\nabla_\lambda J(\lambda) = \sum_{i=1}^{N} (2y^{(i)} - 1) \cdot \nabla_\lambda P_i,$$

$$\nabla_\lambda \mathcal{L}(\lambda) = \sum_{i=1}^{N} \frac{P_i - y^{(i)}}{P_i(1-P_i)} \cdot \nabla_\lambda P_i.$$

Let's analyze the coefficients of $\nabla_\lambda P_i$ in both expressions:

- If $y^{(i)} = 1$:
  - Coefficient in $\nabla_\lambda J(\lambda) = (2y^{(i)} - 1) = (2 \cdot 1 - 1) = 1$.
  - Coefficient in $-\nabla_\lambda \mathcal{L}(\lambda) = \frac{y^{(i)} - P_i}{P_i(1-P_i)} = \frac{1-P_i}{P_i(1-P_i)} = \frac{1}{P_i}$ (for $P_i \neq 1$).

  Both are positive for $P_i \in (0, 1)$, pushing λ towards increasing P_i.
- If $y^{(i)} = 0$:
  - Coefficient in $\nabla_\lambda J(\lambda) = (2y^{(i)} - 1) = (2 \cdot 0 - 1) = -1$.
  - Coefficient in $-\nabla_\lambda \mathcal{L}(\lambda) = \frac{y^{(i)} - P_i}{P_i(1-P_i)} = \frac{0-P_i}{P_i(1-P_i)} = -\frac{1}{1-P_i}$ (for $P_i \neq 0$).

  Both are negative for $P_i \in (0, 1)$, pushing λ towards decreasing P_i.

Despite the different weighting of the linear combinations of the vectors $\nabla_\lambda P_i$, the coefficient always has the same sign. This confirms that both optimization objectives are fundamentally pushing the parameters λ in directions that aim to increase the success probability P_i for positive queries and decrease it for negative queries.

□

B MDP Implementation

B.1 Memory-enhanced Environment

To enable effective reinforcement learning and prevent loops during derivation, we augment the environment with a memory mechanism that tracks visited goals within a single derivation episode. At each step, the environment excludes any action (clause) that would lead to a previously encountered goal, ensuring that the agent explores only new goals.

B.2 False Action

We extend the action space with a special *False* action, which leads to the *False* terminal goal. This serves two purposes: (i) to ensure the agent remains probabilistic even when only one clause is unifiable, and (ii) to enable learning when to abandon early.

For example, consider a logic program $\mathcal{P} = \{C_1, C_2, C_3\}$, where $C_1 = \textit{locIn}(X, Y) \leftarrow \textit{neighOf}(X, Z), \textit{locIn}(Z, Y)$, $C_2 = \leftarrow \textit{neighOf}(tr, gr)$, and $C_3 = \leftarrow \textit{locIn}(gr, eu)$. Consider a negative query $q = G_0 = \leftarrow \textit{locIn}(tr, eu)$ with label $y = 0$. The only unifiable clause is C_1, resulting in a deterministic step. To preserve stochastic behavior, we add *False* action to the action space: $\mathcal{A}(G_0) = \{(C_1, \{X/tr, Z/eu\}), \textit{False}\}$. This allows the agent to distribute probability mass across multiple actions, including the option to terminate early by selecting *False*.

C Experiments

To ensure reproducibility across all experiments, we set the random seeds for all relevant libraries and components. Specifically, we initialize the seed for Python's built-in random module, NumPy, and PyTorch. For GPU computations, we set the seed for CUDA's random number generators on all devices. Additionally, we configure PyTorch's backend settings by enabling deterministic operations and disabling benchmarking to prevent nondeterministic optimizations during training.

C.1 MNIST Addition

Dataset. We follow the same data generation process for the MNIST addition experiment as in previous works (Manhaeve et al. 2018). This involves consecutively selecting $2N$ images from the MNIST dataset and splitting them into two distinct sequences, each of length N. To create supervision signals, we compute the target sum values by multiplying the labels of the selected MNIST images by the corresponding powers of 10 and summing the resulting sequences. Finally, the two numbers are added together to form the final label. Each MNIST image is used only once in all sequences, resulting in $\lfloor 60000/2N \rfloor$ learning samples. They are further split in a 5:1 ratio for training and validation. The test set is generated similarly using the MNIST test partition.

Neural Network. As in previous works, we use a standard LeNet architecture in all MNIST addition experiments to output the probability distribution over digits, indicating the likelihood that each image in the two sequences corresponds to a specific digit (Manhaeve et al. 2018).

Dynamic Programming (DP). We adopt a structured factorization of the problem as digit-wise addition with carry propagation, similar to DeepSoftLog (Maene and De Raedt 2023). This formulation scales linearly in the sequence length N. During DP, we maintain two tables: one indexed by (i, s_i) and another by (i, c_i), where $i \in \{0, \ldots, N-1\}$ denotes the digit position (with $i = 0$ being the least significant digit), s_i is the observed sum digit at position i, and c_i is the carry propagated from position i to $i + 1$. Each entry in the DP tables represents the total probability mass of all valid latent explanations (i.e., digit assignments) leading to that state. The probability computation accounts for the distribution over the incoming carry from the previous digit, as well as the output distributions of the neural digit classifier for the two MNIST images at the current position. Since we train only on positive queries in this problem, maximizing the value function corresponds to maximizing the probability of the ground-truth sum. This probability is computed as the product of the probabilities assigned to the correct sum digit at each position.

The recursive structure of MNIST addition can be represented by the following logic program:

```
mnist_addition([], [], [], 0).
mnist_addition([], [], [1], 1).
mnist_addition([HN1|TN1], [HN2|TN2],[HSUM|TSUM], CarryIn) :-
    Sum is HN1+HN2+CarryIn,
    HSUM is Sum mod 10,
    CarryOut is Sum // 10,
    mnist_addition(TN1, TN2, TSUM, CarryOut).
```

In the recursive clause, the head represents the current goal. For a given digit pair (100 combinations in total), the only non-deterministic atom in the body is the recursive call advancing to the next digit position. As a result, the goal space grows linearly with N. This structure provides a general insight for neurosymbolic problem representations: if a problem can be encoded as a recursive formula in which the only non-deterministic atom is the next goal and the number of substitutions is tractable, then the problem can be solved efficiently using DP.

Policy Gradient. The logic program for masking out invalid actions is shown below. In each training iteration, we sample rollouts that satisfy the ground-truth sum from the masked distribution. These rollouts are used to update the unmasked policy in an off-policy manner, with importance sampling applied to correct for the distribution mismatch.

```
% Generate a mask list of 10 entries (0-9)
digit_mask(Pos, SeqLen, SumDigit, FullSum, CurrNo, Prev, Mask) :-
    findall(Flag, (
        between(0, 9, D),
        ( valid_digit(Pos, SeqLen, SumDigit, FullSum, CurrNo, Prev, D) ->
            Flag = 1
        ;
            Flag = 0
        )
    ), Mask).

% Case: last position, first number
valid_digit(Pos, SeqLen, SumDigit, _, 0, _, D) :-
    Pos =:= SeqLen - 1,
    Remain is SumDigit - D,
    between(0, 9, Remain).

% Case: not last, first number
valid_digit(Pos, SeqLen, SumDigit, FullSum, 0, _, D) :-
    Pos < SeqLen - 1,
    Remain is SumDigit - D,
    RemainLen is SeqLen - Pos - 1,
    max_suffix(FullSum, RemainLen, MaxSuffix),
    (MaxSuffix -> MaxDiff = 9 ; MaxDiff = 10),
    between(0, MaxDiff, Remain).

% Case: last position, second number
valid_digit(Pos, SeqLen, SumDigit, _, 1, Prev, D) :-
    Pos =:= SeqLen - 1,
    Pred is Prev + D,
    Pred =:= SumDigit.

% Case: not last, second number
valid_digit(Pos, SeqLen, SumDigit, FullSum, 1, Prev, D) :-
    Pos < SeqLen - 1,
    Pred is Prev + D,
    RemainLen is SeqLen - Pos - 1,
    max_suffix(FullSum, RemainLen, MaxSuffix),
    ( MaxSuffix -> Pred =:= SumDigit ;
    PredSum =:= SumTarget - 1 ).

% Helper predicates
max_suffix(FullSum, RemainLen, true) :-
    pow10(R, RemainLen),
    FullSum mod R =:= R - 1, !.
max_suffix(_, _, false).
pow10(1, 0).
pow10(Result, N) :-
    N > 0,
    N1 is N - 1,
    pow10(R1, N1),
    Result is R1 * 10.
```

Computation Resources. We conduct our experiments on a workstation equipped with 2*AMD EPYC 7502 CPUs @ 2.5 GHz, 256 GB RAM, and 4*NVIDIA RTX A5000 GPUs (24GB each). All experiments are executed on a single GPU. Since training time can be significantly affected by the current load on the workstation, we report a normalized training time for fair comparison. Specifically, we estimate total training time by measuring the duration of a single epoch on a laptop with an Intel Core i7-10850H CPU @ 2.70GHz, 32GB RAM, and an NVIDIA GeForce RTX 2070 with Max-Q Design (8GB), and multiplying it by the number of epochs required for convergence on the workstation.

Hyperparameters The optimal hyperparameters for DPrL(DP), DPrL(PG), and Scallop were determined via grid search on a held-out validation set. We evaluated learning rates $\{0.0003, 0.001, 0.003\}$ and batch sizes $\{32, 64, 128\}$. For DPrL(DP), the maximum number of training epochs was set to 5000 for all values of N. The best results for $N = 4, 15, 200$ were achieved with a learning rate of 0.003 and batch size 128, while for $N = 100$ and 500, the optimal configuration was a learning rate of 0.001 and batch size 64. For DPrL(PG), the maximum number of training epochs was set to 10,000 for all N. Experiments with $N = 4, 15$ performed best using a learning rate of 0.001 and batch size 32. For $N = 100, 200, 500$, we report performance using a learning rate of 0.003 and batch size 32. For Scallop, the maximum number of training epochs was set at 20 across all N. Due to its top-k approximate inference, selecting k is critical to balance accuracy and computational efficiency. We explored $k \in \{1, 3, 5, 6, 7\}$ based on the original Scallop implementation. In all cases where Scallop did not time out, $k = 3$ with a learning rate of

0.001 and batch size 32 yielded state-of-the-art performance and the fastest runtimes. Each experiment was run with 5 different random seeds.

For KLay (Maene, Derkinderen, and Martires 2025), DeepStochLog (Winters et al. 2022), DeepSoftLog (Maene and De Raedt 2023), A-NeSI (van Krieken et al. 2023), and EXAL (Verreet et al. 2024), we used their best reported accuracies. The training times were obtained with the optimal hyperparameters provided in their respective papers.

C.2 Explainable Knowledge Graph Completion

Detailed dataset statistics The statistics for the datasets used are provided in Table 4.

Dataset	Entities	Rels.	Facts	Avg. Degree	Rules
Family	3,007	12	19,845	6.47	48
WN18RR	40,559	11	86,835	2.14	28

Table 4: Detailed dataset statistics.

Evaluation Metrics for KG Completion To assess model performance in the link prediction task, we employ three standard ranking metrics:

Mean Reciprocal Rank (MRR). This metric calculates the average reciprocal of the rank of the first correct entity for a set of test queries Q:

$$\text{MRR} = \frac{1}{|Q|} \sum_{q \in Q} \frac{1}{\text{rank}_q}$$

where rank_q is the rank of the correct entity for query q.

Hits@N. This metric measures the proportion of queries for which a correct entity appears in the top-N ranked predictions. It is calculated as:

$$\text{Hits@N} = \frac{1}{|Q|} \sum_{q \in Q} \mathbb{I}(\text{rank}_q \leq N)$$

where $\mathbb{I}(\cdot)$ is the indicator function.

Area Under the Precision-Recall Curve (AUC-PR). This metric computes the area under the Precision-Recall curve, which plots precision against recall at various decision thresholds. AUC-PR is particularly informative for imbalanced datasets, common in link prediction, as it focuses on the performance of positive instances.

Computation Resources. We used an Intel Core i7 CPU @ 2.10 GHz, 48 GB of RAM, 2*NVIDIA Tesla V100-SXM2 GPUs (32 GB each).

Hyperparameters. For the knowledge graph experiments, due to the cost of running multiple seeds, the seed was fixed to 0. We explored the values of 50-200 for embeddings size, 0.2, 0.5 for entropy coefficient, and 0.1, 0.2 for clip range. For both datasets, the entropy coefficient of PPO was set to 0.2, as well as the clip range. The learning rate is set to 3e-4 and the embedding size 64. The rest of the parameters can be consulted in the repository.