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[4,5], Michelangelo Diligenti[2], Giuseppe Marra[1]

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

Abstract

Neurosymbolic (NeSy) AI combines neural architectures and symbolic reasoning to improve accuracy, interpretability, and generalization. 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.

Code — https://github.com/DeepProofLog/DPrL-AAAI
Extended version — http://arxiv.org/abs/2511.08581

1 Introduction

Neurosymbolic (NeSy) AI represents a promising paradigm that seeks to bridge the gap between data-driven neural architectures and the 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 one 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, their applicability to large-scale relational settings remains out of 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 that takes 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. From 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 parameter-

*Equal first authorship

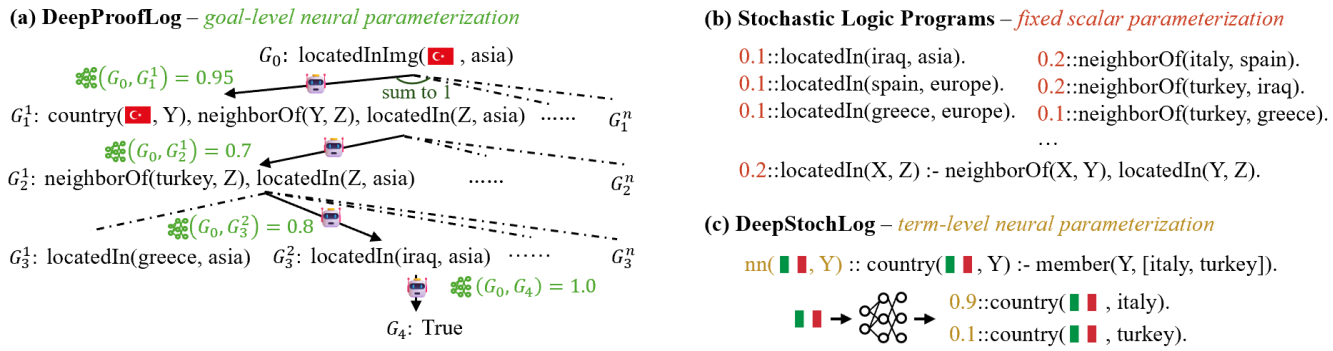

Figure 1: (a) DeepProofLog uses goal-level parameterization: a neural policy conditioned on the goal to guide proving at each resolution step. In comparison, (b) SLPs assign scalar weights to clauses for probabilistic proof search without input adaptation, and (c) DeepStochLog uses neural nets conditioned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

ized 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 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 (RL) to approximate the optimal policy through learning from sampled derivations.

While automated theorem proving (ATP) can also be framed as an MDP and solved with RL, NeSy reasoning differs conceptually. ATP aims to find any valid proof, whereas NeSy focuses on probabilistic logics, weighing multiple proofs and assigning low probabilities to those inconsistent with subsymbolic signals such as images or embeddings. Consequently, DPrL must both identify plausible proofs and ensure their probabilities align with these subsymbolic signals—this alignment is the central challenge we address to improve efficiency in NeSy learning.

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. (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) Experiments on NeSy benchmarks and knowledge graph completion tasks empirically validate the effectiveness of our approach, achieving improved task performance, efficiency, and scalability over 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 a *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

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

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 derivation. 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 end, we introduce DeepProofLog, which defines clause selection probabilities conditioned on the entire current goal and enables dynamic adaptation to varying 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:

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

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 the 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 are 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 of 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 as 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 on 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, 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 the 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: $C_1 = \textit{locIn}(X, Z) \leftarrow \textit{neighOf}(X, Y), \textit{locIn}(Y, Z)$, $C_2 = \leftarrow \textit{neighOf}(\textit{it}, \textit{fr})$, $C_3 = \leftarrow \textit{locIn}(\textit{fr}, \textit{eu})$, $C_4 = \leftarrow \textit{locIn}(\textit{tr}, \textit{gr})$, 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. For the empirical distribution defined by $\mathcal{D}$, the objective is defined as $J(\pi_\lambda) = \sum_{i=1}^{N} [V^{\pi_\lambda}(q^{(i)}, y^{(i)})]$ (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 for future work.

6 Experiments

We present two sets of experiments. First, we assess our approach on a neurosymbolic task that learns 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 such as 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's 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 on predicted sum accuracy and training time. 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). Results are summarized in Table 1, with experimental setups, hyperparameters, and additional intermediate N values 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 computa-

	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.

tional efficiency, but converges to a suboptimal policy due to the vast combinatorial search space.

6.2 Explainable Knowledge Graph Completion

We evaluate DPrL on popular KG completion benchmarks: Family (Kok and Domingos 2007) and WN18RR (Dettmers et al. 2018). Family models familial relationships, while WN18RR is a challenging subset of WordNet with inverse relations removed (dataset statistics in Appendix C.2). All methods were tested with 200 randomly sampled negative corruptions per query.

Table 2 compares DPrL to KGE baselines (ComplEx (Trouillon et al. 2016), RotatE (Sun et al. 2019)) and logic-based NeSy systems: 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. 2025) We indicate these variants as *R2N*$_{\text{SG}}$, *DCR*$_{\text{SG}}$, and *SBR*$_{\text{SG}}$. However, these fast grounding techniques require manual parameter tuning, 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.

On Family, all NeSy systems improve performance over the baseline. DPrL is particularly effective, especially for MRR and Hits@1. A similar trend is observed on WN18RR, where DPrL and SBR outperform other methods.

Unlike the partially latent reasoning done by R2N or DCR, DPrL is fully interpretable. Figure 2 presents two proof trees generated by DPrL: a depth-3 proof for synset_domain_topic_of(09788237, 08441203) from WN18RR, and a proof for uncle(2266, 2252) from Family.

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 a high depth. In contrast, DPrL can avoid an exhaustive search by

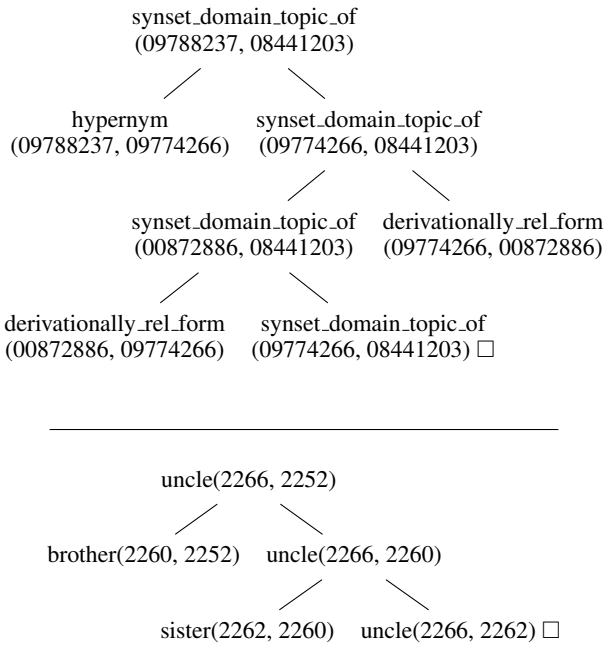

Figure 2: Examples of multi-step proof trees discovered by DeepProofLog for the Family and WN18RR datasets.

learning an efficient proof-finding policy, which navigates deep reasoning paths efficiently. For instance, on the test set of Family, the smart grounding methods of (Ontiveros et al. 2025) instantiate 18,071, 45,466, and 181,095 groundings at depths 1, 2, and 3, respectively. In contrast, DPrL needs just 12,334 ground evaluations.

In conclusion, DPrL demonstrates superior scalability over other perception- and structure-learning NeSy models, such as DeepSoftLog (which fails to scale to the datasets presented here), while preserving the explainability and interpretability advantages of logic-based NeSy approaches.

7 Related Work

DeepStochLog. DeepStochLog extends SLPs by parameterizing clause probabilities via neural networks 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, DPrL 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-parallel 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 provided by DPrL.

Automated Theorem Proving. Automated Theorem Proving (ATP) finds a proof for a query within deterministic logical frameworks. Recent advances 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, DPrL 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 (DPrL), a scalable and flexible NeSy framework that bridges expressive logical representations and efficient data-driven learning. DPrL 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 dynamic programming and RL 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).

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), the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (Grant agreement No. 101142702), and the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme. 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. M. Diligenti and M. Gori are also supported by the project "CONSTR: a COllectionless-based Neuro-Symbolic Theory for learning and Reasoning", PARTENARIATO ESTESO "Future Artificial Intelligence Research - FAIR", SPOKE 1 "Human-Centered AI" Università di Pisa, "NextGenerationEU", CUP I53C22001380006. G. Marra acknowledges support by FWO (G033625N) and KU Leuven Research Fund (CELSA, CELSA/24/008).

Y. Jiao would like 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.