AN ARCHITECTURE FOR PREDICATE LEARNING AND ITS USES IN PLANNING

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce a new architecture for learning unary and binary predicates from positive and negative examples over relational structures with a fixed set of primitive predicates. The architecture, called Neural Description Logic Machine or NDLM, is based on a rich description logic which has the expressive power of 3-variable first-order logic, FO^3. The architecture is restricted to predicates of arity one and two, and unlike related architectures such as neural logic machines (NLMs), edge-transformers, and (oblivious) 3-GNNs, it achieves this expressivity requiring quadratic rather than cubic space. The predicate learning task considered is a generalization and variation of inductive logical tasks such as concept learning and inductive logic programming but where the representation of the learned predicates is not given in terms of a first-order formula or logic program. Still we show that the learned, neural representation of concepts and roles is not fully opaque either, and in many cases, can be understood logically as well. We also show that a number of tasks in "learning for planning" can be reduced to predicate learning tasks of this form, including lifted STRIPS model learning from state-action traces, and general policy learning from plans. The performance of NDLMs is evaluated experimentally over a number of planning and concept learning tasks where it is compared with NLMs and state-of-the-art symbolic and neural unary and binary-relational learners.

1 INTRODUCTION

The problem of learning relations from examples is pervasive. In the symbolic setting, it is formulated and addressed in the context of inductive logic programming (Muggleton & De Raedt, 1994; Cropper & Morel, 2021), concept learning (Lehmann & Hitzler, 2010; Funk et al., 2025), and conjunctive query synthesis (Thakkar et al., 2021; Cate & Dalmau, 2022). In all these cases, the task is to learn a formula that defines a target relation in terms of primitive relations. In the setting of planning, and more precisely, generalized planning, unary predicates have been learned without supervision for generating general policies; namely, policies that generalize across entire planning domains (Frances et al., 2021; Drexler et al., 2022). In the neural setting, similar problems including concept learning and link prediction have been addressed using different architectures, including graph neural networks (GNNs) (Scarselli et al., 2009), higher-order GNNs (Morris et al., 2019), graph transformers (Bergen et al., 2021), and neuro-symbolic architectures among others (Demir & Ngomo, 2023).

The strength of symbolic methods is that they are transparent, precise, and interpretable, and don't need many examples to learn. Their weakness is that they do not scale up well. Neural methods have the reverse strengths and weaknesses: they scale up well, but the concepts learned do not generalize equally well, they require more data to learn, and they can't be interpreted easily.

The aim of this work is to bring symbolic and neural approaches closer together by introducing a new deep-learning architecture for learning unary and binary predicates from positive and negative examples over relational structures with a fixed set of predicates. The architecture, called Neural Description Logic Machine or NDLM, is based on a rich description logic which has the expressive power of 3-variable first-order logic, FO^3. The architecture is closely related to neural logic machines or NLMs (Dong et al., 2019) which are based on logic programs. Like edge-transformers (Bergen

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

et al., 2021), and (oblivious) 3-GNNs (Morris et al., 2019), which even capture FO^3 with counting quantifiers, NLMs need cubic space for this expressivity, while NDLMs need only quadratic space.

2 The Task: Predicate Learning

Predicate learning is the task of learning a new predicate from a given set of predicates P and labeled examples. For example, there may be a database of family relations containing *parent*(x, y) and *brother*(x, y) and positive and negative examples of (direct) *uncle*(x, y). The goal is to learn a formula such as $\Phi(x, y) := \exists z(\textit{brother}(x, z) \wedge \textit{parent}(z, y))$. This kind of learning problem appears in different contexts, for example, in inductive logic programming (ILP) (Muggleton & De Raedt, 1994), where the function is represented as a logic program, in concept learning (Lehmann & Hitzler, 2010), where the function is a description logic formula, and in conjunctive query synthesis (ten Cate & Dalmau, 2015), where the new relation is based on a conjunction of lifted atoms.

In this work, as in other neural approaches, the target function is represented not as a logic formula but by the parameters of a neural network. The input to the neural network is given as ground atoms $p(o_1, \ldots, o_k)$ of predicates $p \in P$, where k is the arity of the predicate and o_i is an object. While in this work, only unary and binary predicates are considered, the predicate learning problem is formulated for the general case. An interpretation $\mathcal{I}$ defines a set of objects $\Delta^{\mathcal{I}}$ as well as an interpretation function that maps a predicate p of arity k to a set of groundings $p^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}})^k$ that define the *extension* of the predicate in the interpretation. Equivalently, and more conveniently, an interpretation $\mathcal{I}$ can be defined as a set $S^{\mathcal{I}}$ of ground atoms: those in the set are the true ground atoms, those not in the set are the false ground atoms.

In this learning setting each example is a pair $(\mathcal{I}_j, t^{\mathcal{I}_j})$ where t is the target predicate, and $\mathcal{I}_j$ is a full interpretation over the set of input predicates P and a *partial* interpretation $t^{\mathcal{I}_j}$ over the target predicate t, $t^{\mathcal{I}} : (\Delta^{\mathcal{I}})^k \rightharpoonup \{true, false\}$. Thus, for the input predicates a closed world is assumed, while for the target predicate an open world is assumed instead. This is because, the examples for the target predicate t are not assumed to place *every* tuple of ground arguments in either the extension of t or its negation. For convenience, we will sometimes write $p(o)$ instead of $p(o_1, \ldots, o_k)$ where o denotes the tuple of constants $o_1, \ldots, o_k$.

Since object names carry no information by themselves, the target function f must depend only on the relational structure, that is, it must be equivariant: for every isomorphism $\pi : \mathcal{I} \to \mathcal{J}$ between interpretations of P and every $o \in (\Delta^{\mathcal{I}})^k$, it holds that $o \in f^{\mathcal{I}}$ if and only if $\pi(o) \in f^{\mathcal{J}}$.

Definition 2.1 (Predicate Learning)**.** Given training examples $E = \{(\mathcal{I}_1, t^{\mathcal{I}_1}), \ldots, (\mathcal{I}_n, t^{\mathcal{I}_n})\}$, where $t^{\mathcal{I}_i}$ is a (partial) interpretation of the k-ary target predicate t given the interpretation $\mathcal{I}_i$, the (semantic) predicate learning task is to find an equivariant, k-ary target function $f^t_{P,E}(\mathcal{I})$ that is consistent with E and maps every finite interpretation $\mathcal{I}$ of P into a full interpretation $t^{\mathcal{I}}$, where $t^{\mathcal{I}} = f^t_{P,E}(\mathcal{I}) \subseteq (\Delta^{\mathcal{I}})^k$. The function is consistent with E if for all $\mathcal{I}_i$ and all $o \in (\Delta^{\mathcal{I}_i})^k$ it must hold that 1) if $t^{\mathcal{I}_i}(o) = \textit{true}$, then $o \in f^t_{P,E}(\mathcal{I}_i)$ and 2) if $t^{\mathcal{I}_i}(o) = \textit{false}$, then $o \notin f^t_{P,E}(\mathcal{I}_i)$.

The learned function $f^t_{P,E}(\mathcal{I})$ does not have to be a symbolic function and can be captured by a neural network. Still the function may be defined by a logical formula $\Phi_{P,E}(x)$ where x is the tuple of free variables of the same arity as t. In such a case, $f^t_{P,E}(\mathcal{I})$ stands for the set of object tuples o in $\mathcal{I}$ of the same arity as x such that $\Phi_{P,E}(o)$ is true in $\mathcal{I}$.

3 Applications in Planning

There have been many applications of predicate learning in the setting of graph classification and also to knowledge graphs, where unary predicate learning relates to concept learning, and binary to (inductive) link prediction (Morris et al., 2019; Demir et al., 2025; Zhu et al., 2021). We will evaluate our NDLMs on such benchmarks for calibration, yet we are particularly interested in novel applications of predicate learning in the setting of planning, where 1) predictive action models are to be learned from suitable state-action traces, and 2) general policies are to be learned from plans.

3.1 ACTION MODEL LEARNING

An example is learning the action model of the sliding-tile puzzle, or n-puzzle, from traces $s_0, a_0, s_1, a_1, \ldots$ where s_i is a state (a set of ground atoms over predicates *at-tile*, *at-blank*, *right-of*, and *above-of*), and a_i is one of the actions *up*(), *down*(), *left*(), and *right*(). Only a few action-model learners, like SYNTH, learn from traces in which not all action arguments are explicit (Jansen et al., 2025). For instance, the action *right*(), that swaps the blank with the tile to its right, is applicable if and only if the *blank* is not in the rightmost column. Predicting positive and negative effects on atoms *at-tile*(t, c) in the resulting state, where these atoms encode the tile positions, can indeed be cast as a predicate learning task with target predicates *at-tile*$^+(x, y)$ and *at-tile*$^-(x, y)$, from the interpretations encoded in the state s and the action a. Logically, the action *right*() has the effect *at-tile*(x, y) if and only if the formula *at-blank*$(y) \wedge \exists z($*right-of*$(z, y) \wedge$ *at-tile*$(x, z))$ is true in s.

Formally, a STRIPS planning domain consists of predicates P and action schemas $a(x_1, \ldots, x_n)$, which have preconditions *Pre*(a), delete effects *Del*(a), and add effects *Add*(a), each of which is a set of lifted atoms over P using variables x_i (Russell, 2010). An instance defines a set of objects O, and a state s is a set of ground atoms over these objects. An action $a(o_1, \ldots, o_n)$ is applicable in s if s contains all atoms $p(x)$ in *Pre*$(a(o))$ where each x_i is replaced by o_i, and has effects defined by *Add*$(a(o))$ and *Del*$(a(o))$, which are grounded the same way, resulting in the state $s' = (s \setminus Del(a(o))) \cup Add(a(o))$. While in STRIPS every variable contained in any of the lists *Pre*(a), *Add*(a), and *Del*(a) has to be explicit in the action arguments, in STRIPS+, arguments can be left implicit if they can be uniquely determined in the preconditions (Jansen et al., 2025).[1]

Each schema a is learned separately from observed transitions $(s, a(o), s')$ and pairs $(s, a(o))$ where $a(o)$ is not applicable in s. Every positive and negative example yields an interpretation with $\Delta = O$ and predicates $P \cup \{A_1, \ldots, A_n\}$, where A_i is only true for the i-th argument o_i of the applied action $a(o)$. Preconditions are learned of an action a for a state s using a single target predicate $t^{a,s}$, where $o \in t^{a,s}$ if and only if the action $a(o)$ is applicable in s. For positive examples and each predicate $p \in P$, the target predicates $t^{p+} = \{o \mid p(o) \notin s \wedge p(o) \in s'\}$ and $t^{p-} = \{o \mid p(o) \in s \wedge p(o) \notin s'\}$, contain all atoms that become true (added) and that become false (deleted), respectively.

3.2 GENERAL POLICY LEARNING

Consider a domain where an agent is at some position c of a grid and its goal is to be in another cell c', and the agent can move one unit at a time in each direction. A simple general policy for this domain must map each state s to an action that moves the agent closer to the goal.

Formally, a general policy for a domain maps each state of *each problem* to a set of actions, and the policy solves the domain if, for any problem, repeatedly applying any action from this set, eventually reaches the problem goal (Frances et al., 2021). Given a planning problem in STRIPS+, we consider pairs $(s \cup g, a(o))$ of examples, positive if $a(o)$ is an optimal action in the state s for the goal g of some domain instance, and negative if it is not. For each action schema, a predicate learning problem can then be defined with the action schema name a as the target predicate t. We want to find out which groundings of the lifted action $a(x)$, if any, are "optimal" in the interpretations defined by pairs of states s and goals g that are not in the examples and may belong to different domain instances (Ståhlberg et al., 2022). Indeed, each state s with goal g yields an interpretation with domain O over the domain predicates p and their "goal-copies" p_g, where the goal atom $p_g(o)$ is true if it is in g. For the experiments below, however, we don't test if the learned "relations" $a(o)$ express that actions $a(o)$ are optimal for achieving g from s, but whether their execution leads to g, optimally or not. If they do, the learned "predicates" a, one for each action schema name a, represent a general policy for the domain.

4 RELATED WORK

We review symbolic and neural approaches briefly; more details can be found in the Appendix E.

[1] For example, in STRIPS+, the n-puzzle can be modeled as above with no arguments, while in plain STRIPS, three arguments each are required, for the blank position, the next blank position, and the tile in such position.

Symbolic approaches. In symbolic approaches, the function to be learned for the target predicate is expressed as a formula in terms of the input predicates. The logic and the syntactic form of these formulas vary. In *concept learning* (Lehmann & Hitzler, 2010), the target predicate is unary, and the logic is a description logic where unary predicates are called concepts, and binary predicates, roles. The algorithms that search for the target formulas are mostly based on heuristic search, sometimes with neurally trained heuristics (Demir & Ngomo, 2023), or SAT (Funk et al., 2025). In *inductive logic programming* (Muggleton & De Raedt, 1994; Cropper & Morel, 2021), the target formula is captured by a logic program, while in *conjunctive query fitting* (ten Cate & Dalmau, 2015), it is a conjunction of atoms with a number of free variables equal to the arity of the target predicate and other variables quantified existentially.

Neural approaches. Neural logic machines (NLMs) (Dong et al., 2019) are a neurosymbolic architecture for learning relations from data in a form that mimics logic programs and ILP. Our Neural description logic machines (NDLMs) are based instead on description logic. The main practical difference between the two, however, is the use of memory. NDLMs capture the 3-variable fragment of first-order logic over unary and binary relations using quadratic space in the number of objects, while NLMs require cubic space. Plain GNNs, on the other hand, have the ability to capture the 2-variable fragment with counting quantifiers, while k-GNNs for $k > 2$, capture the k-variable fragment using space exponential in k (Morris et al., 2019; Xu et al., 2019). Edge-transformers (Bergen et al., 2021) have the power of 3-GNNs with the same space complexity, while PPGNs (Maron et al., 2020) reduce this complexity to quadratic.

5 Architecture

Neural description logic machines (NDLMs) learn concepts (unary predicates) and roles (binary predicates) mimicking a description logic whose expressive power matches FO^3 (details in Appendix D.1). Below, we introduce the grammar, define the architecture, state its properties, and describe the training procedure. NDLMs are composed of several layers such that each layer maps a number of unary and binary relations in the input, represented parametrically, into a number of new unary and binary relations in the output.

5.1 The Description Logic in NDLMs

NDLMs are based on the description logic $\mathcal{D} = \mathcal{ALC}(\neg, \sqcap, \sqcup, ^-, \circ, id, \triangleright)$, that is the base logic $\mathcal{ALC}$ extended with Boolean operations on roles, role inverse, role composition, the identity role, and domain restriction (Baader et al., 2007). The grammar for the concepts C and roles R is:

$$\begin{aligned} C &::= A \mid \top \mid \bot \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists R.C \mid \forall R.C, \\ R &::= P \mid \top_R \mid \bot_R \mid id \mid \neg R \mid R \sqcap S \mid R \sqcup S \mid R^- \mid R \circ S \mid C \triangleright R. \end{aligned} \tag{1}$$

Interpretations map concepts to subsets of $\Delta^{\mathcal{I}}$ and roles to subsets of $(\Delta^{\mathcal{I}})^2$. The operators have the usual semantics, which can be found in appendix, and unlike the base logic $\mathcal{ALC}$ and variants, they can express three-variable formulas. The architecture exploits that some grammar rules are truly relational, while others are Boolean. In particular, the grammar rules involving the concepts $\exists R.C$ and $\forall R.C$, and the roles R^-, $R \circ S$, and $C \triangleright R$, all involve "relational" operations, while the other rules can be captured "propositionally".

5.2 Layers

Each of the $\mathbb{N}$ layers of an NDLM, transforms the input concepts and roles to the layer, $C \in \mathbb{R}^{c \times n}$ and $R \in \mathbb{R}^{r \times n \times n}$, into output concepts and roles, where n is the number of objects, and c and r are the number of input concept and roles. A layer of an NDLM is displayed in Figure 1. Every layer works in two stages. The first stage generates intermediate concepts and roles in a fixed way, by applying the "relational" grammar rules once to the input concepts and roles. This stage does not contain any learnable parameters. The second stage consists of two independent parts: one that combines the intermediate concepts into new concepts using a feed-forward neural network (FFN), and one that does the same for roles. Both compute soft, learnable Boolean combinations that apply in the same way to all objects and all pairs of objects, respectively.

A NDLM can learn multiple target predicates, and the final layer produces a task-dependent number of output concepts c_{out} and roles r_{out}. A nullary target is represented by an output concept whose target value is 1 for all objects if the target is true and 0 for all objects if it is false.

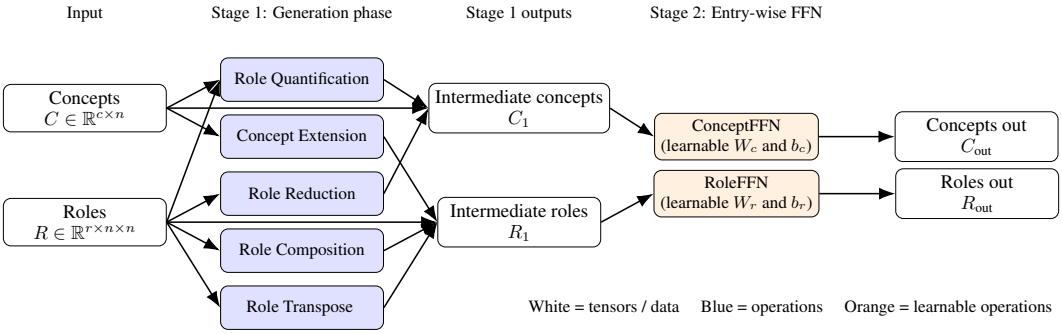


Figure 1: A single layer of a Neural Description Logic Machine (NDLM).

5.2.1 Generating Intermediate Concepts and Roles

To generate intermediate concepts and roles, NDLMs use a matrix-product-like operation where the sum is replaced with $\min$ and $\max$ aggregations. Let $\mathcal{A} = \{\min, \max\}$ denote the aggregators; we define the generalized matrix product for matrices $A \in \mathbb{R}^{a \times n}$ and $B \in \mathbb{R}^{n \times b}$ with $\alpha \in \mathcal{A}$ as

$$(A \star_\alpha B)_{p,q} := \underset{l \in [n]}{\alpha} A_{p,l} B_{l,q} \qquad \text{for all } p \in [a],\ q \in [b].$$

In the following, $\otimes$ denotes the concatenation of the results obtained from the aggregators in $\mathcal{A}$.

Role Quantification (RQ) aggregates every concept along the outgoing edges of every role:

$$\mathrm{RQ}\colon\ \mathbb{R}^{c \times n} \times \mathbb{R}^{r \times n \times n} \to \mathbb{R}^{2rc \times n}, \qquad (C, R) \mapsto \bigotimes_{\alpha \in \mathcal{A}} \left(R_i \star_\alpha C_j\right)_{(i,j) \in [r] \times [c]}.$$

This operation is closely related to quantification along roles in description logic. In particular, for $\max$ aggregation and Boolean inputs, it is equivalent to $\exists R_i.C_j$.

Role Composition (RC) combines all ordered pairs of roles:

$$\mathrm{RC}\colon\ \mathbb{R}^{r \times n \times n} \to \mathbb{R}^{2r^2 \times n \times n}, \qquad R \mapsto \bigotimes_{\alpha \in \mathcal{A}} \left(R_i \star_\alpha R_j\right)_{(i,j) \in [r] \times [r]}.$$

For a pair of objects $p, q \in [n]$, this is the largest $(\max)$ or smallest $(\min)$ weight of a path through an intermediate object $l \in [n]$. For $\max$ aggregation and Boolean inputs, this is equivalent to $R_i \circ R_j$.

Role Reduction (RR) derives a concept for each role by aggregation along the first dimension:

$$\mathrm{RR}\colon\ \mathbb{R}^{r \times n \times n} \to \mathbb{R}^{2r \times n}, \qquad R \mapsto \bigotimes_{\alpha \in \mathcal{A}} \left(R_i^\top \star_\alpha \mathbf{1}_n\right)_{i \in [r]}.$$

This operation generates a concept for each role by aggregating over the ingoing edges, where for each object it takes the value of the strongest or weakest ingoing edge. This computes $\exists R_i^-.\top$ for Boolean inputs and $\max$ aggregation.

Concept Extension (CE) turns concepts into roles by repeating the vector of the concept n times:

$$\mathrm{CE}\colon\ \mathbb{R}^{c \times n} \to \mathbb{R}^{c \times n \times n}, \qquad C \mapsto E \text{ such that } E_{i,p,q} = C_{i,p} \text{ for all } i \in [c],\ p, q \in [n].$$

Under the assumption of Boolean values, this corresponds to a role that has an edge (p, q) if and only if the node p is contained in the concept, that is, it computes $C_i \triangleright \top_R$. In a continuous formulation, the edge (p, q) is weighted by the membership weight of p in the concept.

Role Transpose (RT) transposes each role, which reverses the direction of all its edges:

$$\mathrm{RT}\colon\ \mathbb{R}^{r \times n \times n} \to \mathbb{R}^{r \times n \times n}, \qquad R \mapsto \left(R_i^\mathsf{T}\right)_{i \in [r]}.$$

This operation corresponds to the inverse role R_i^- in description logics.

Given c input concepts and r input roles, we concatenate the new predicates with the input predicates, which act as skip connections, and obtain intermediate concepts C' and roles R'. Formally,

$$C' = C \otimes \mathrm{RQ}(C, R) \otimes \mathrm{RR}(R) \qquad \in \mathbb{R}^{c' \times n}, \qquad c' = c + 2r(c+1),$$
$$R' = R \otimes \mathrm{RC}(R) \otimes \mathrm{RT}(R) \otimes \mathrm{CE}(C) \qquad \in \mathbb{R}^{r' \times n \times n}, \qquad r' = 2r + 2r^2 + c.$$

Extension: Transitive Closure Optionally, the transitive closure (TC) can be added to the operators of the first stage. In this case, the transitive closure gives the maximal capacity of any path from p to q of a role. A formal definition can be found in the Appendix Adding these predicates to the intermediate roles allows the downstream FFNs to exploit long-range connectivity information. Because this operation adds $O(n^4)$ complexity, it is only used when explicitly indicated (see Appendix C).

5.2.2 Combining Intermediate Predicates

The second stage of a layer combines the intermediate concepts C' and roles R' into c'' output concepts and r'' output roles. Here, c'' and r'' equal the width m for hidden layers and the number of output predicates for the output layer. The concepts are combined by a $\textsc{ConceptFFN}^l \colon \mathbb{R}^{c'} \to \mathbb{R}^{c''}$ that is applied object-wise, and the roles are combined by a $\textsc{RoleFFN}^l \colon \mathbb{R}^{r'} \to \mathbb{R}^{r''}$ that is applied object-pair-wise:

$$C''_{:,x} = \textsc{ConceptFFN}^l\big(C'_{:,x}\big) = \sigma(W_C^l\, C'_{:,x} + b_C^l) \qquad \text{for all } x \in [n],$$
$$R''_{:,x,y} = \textsc{RoleFFN}^l\big(R'_{:,x,y}\big) = \sigma(W_R^l\, R'_{:,x,y} + b_R^l) \qquad \text{for all } x, y \in [n].$$

Here, $C'_{:,x} \in \mathbb{R}^{c'}$ contains the values of all intermediate concepts for x, and $R'_{:,x,y} \in \mathbb{R}^{r'}$ contains the values of all intermediate roles for the object pair (x, y). In principle, any feedforward neural network (FFN) architecture can be used but in the experiments, a single layer with sigmoid or identity activation functions σ are used with learnable weight matrices $W_C^l \in \mathbb{R}^{c'' \times c'}$ and $W_R^l \in \mathbb{R}^{r'' \times r'}$, and bias vectors $b_C^l \in \mathbb{R}^{c''}$ and $b_R^l \in \mathbb{R}^{r''}$. Both networks share the same parameters for all (pairs of) objects, so they are equivariant and do not depend on the number of objects n.

5.3 Training NDLMs

An NDLM is trained by optimizing the weight matrices W_i and bias vectors b_i of the ConceptFFN and RoleFFN modules of each layer i using Adam (Kingma & Ba, 2014) and L_2 weigth decay and minimizes masked binary cross-entropy between the target predicate t and the networks output. Let $\hat{C}$ be the output concepts and $\hat{R}$ the output roles and let T_C and T_R be the target predicates that are 0 everywhere, where the target is not known. The masks M_C and M_R remove the loss on unobserved groundings of the target predicates. The loss is formalized as:

$$\mathcal{L}_{\text{combined}} = \mathcal{L}_C + \mathcal{L}_R = \frac{\sum M_C \odot \mathrm{BCE}(T_C, \hat{C})}{\sum M_C} + \frac{\sum M_R \odot \mathrm{BCE}(T_R, \hat{R})}{\sum M_R},$$

where $\odot$ denotes element-wise multiplication, $\mathcal{L}_C$ and $\mathcal{L}_R$ are 0 if there are no target concepts or roles, respectively. The binary cross-entropy is $\mathrm{BCE}(y, z) = -y \log(z) - (1 - y) \log(1 - z)$. At evaluation time, continuous predictions are thresholded at 0 for identity activations and at 0.5 for sigmoid activations, and compared with the target values at the labeled positions to compute task-specific measures such as accuracy and F1 score.

5.4 Complexity

For fixed depth and width m, an NDLM (without TC) has $O(m^2 n^3)$ forward-pass time and $O(m^2 n^2)$ activation memory for n objects. The cubic time is due to RC; all other operations are at most quadratic in n. Unlike architectures that explicitly materialize ternary predicates, NDLMs retain quadratic activation memory. Appendix D.4 gives the channel and parameter counts.

6 EXPRESSIVITY

We analyze the expressivity of NDLMs (without transitive closure) with respect to the description logic $\mathcal{D}$ 5.1 inspired by the `DLplan` library (Drexler et al., 2022). As noted above, $\mathcal{D}$ can be characterized exactly by the three-variable fragment of first-order logic with predicates of arity at most two, denoted $\text{FO}^3_{\leq 2}$. (see Appendix D.1)

For the lower bound, we assume that the pointwise CONCEPTFFN and ROLEFFN modules can realize the required Boolean functions exactly on Boolean inputs. This assumption isolates the structural expressivity of the architecture from the choice of activation function.

Theorem 6.1 (Exact expressivity of NDLMs)**.** Let $\tau \in \mathbb{R}$ be a decision threshold and let interpretations range over a fixed finite signature of primitive concepts and roles. Then:

(*Lower bound*) For every concept formula $F \in \mathcal{D}$, there is an NDLM $f^t_{P,\theta}$ with a single output concept ($k = 1$) or role ($k = 2$) such that for every finite interpretation $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$,

$$F^{\mathcal{I}} = \{x \in \Delta^k \mid f^t_{P,\theta}(\mathcal{I})_x = 1\} = \Delta \setminus \{x \in \Delta^k \mid f^t_{P,\theta}(\mathcal{I})_x = 0\}. \tag{2}$$

(*Upper bound*) For every NDLM $f^t_{P,\theta}$ with a single output concept or role, there is a concept formula $F_{N,\tau} \in \mathcal{D}$ such that for every finite interpretation $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$,

$$F^{\mathcal{I}}_{N,\tau} = \{x \in \Delta^k \mid f^t_{P,\theta}(\mathcal{I})_x \geq \tau\}. \tag{3}$$

The depth of the NDLMs required is bounded by twice the operator depth of the respective $\mathcal{D}$ formulas. The proof can be found in the Appendix D.3.

7 EXPERIMENTS

The experiments assess the performance of NDLMs as a predicate-learning architecture across three applications: graph classification, action model learning, and general policy learning. The main comparisons are designed to examine two questions: first, whether NDLMs are competitive with strong graph neural-network baselines; second, whether they can address challenging planning-derived predicate learning tasks. Further experiments are reported in the Appendix B.

7.1 TUDATASET

Graph classification is a special case of nullary predicate learning given unary predicates for the nodes, and binary predicates for the edges. We evaluate NDLM on graph classification tasks from the TUDataset benchmark suite (Morris et al., 2020; Yanardag & Vishwanathan, 2015) and for comparison NLMs in Table 1. We report a subset of the results here; the full table and further experimental details are in Appendix B.1.

Table 1: Graph classification accuracy. Baseline results taken from He & Cheng (2025).

Method	MUTAG	PTC	PROTEINS	NCI1	NCI109	COLLAB	IMDB-B	IMDB-M
PPGN	90.6	66.2	77.2	83.2	82.2	83.1	73.0	50.4
GIN	89.4	64.6	76.2	82.7	–	80.2	75.1	52.3
1-2-3 GNN	86.1	60.9	75.5	76.2	–	–	74.2	49.5
GFN	90.84	–	76.46	82.7	–	80.4	73.3	51.2
DropGIN	90.4	66.3	76.3	–	–	–	75.7	51.4
CIN	92.7	68.2	77.0	83.6	**84.0**	–	75.6	52.7
GSN	92.2	68.2	76.6	83.5	–	**85.5**	**77.8**	54.3
Order-2 GNN-b	92.0	**72.4**	73.9	80.0	81.0	78.5	72.4	51.0
Order-3 GNN-a	94.4	72.1	77.8	**85.0**	83.4	80.7	74.9	52.5
NLM	**97.22**	67.65	76.58	80.29	80.26	71.80	74.50	**54.67**
NDLM (ours)	88.89	66.18	**80.63**	81.75	80.44	79.30	72.67	**54.67**

NDLM performs particularly well on PROTEINS and IMDB-MULTI, but is weaker on MUTAG, PTC, and IMDB-BINARY. In IMDB-MULTI, both NDLM and NLM rank first among the existing state-of-the-art methods. NDLMs and NLMs use min and max aggregation functions, while the other methods rely on (weighted) sum or mean, thus representing a different tradeoff between expressivity and generalization.

7.2 Action Model Learning

The action model experiments implement the description in Section 3.1. Preconditions and effects are trained as separate predicate-learning problems for each action schema. The experiments use STRIPS+ domains with fewer explicit action arguments than STRIPS domains, where predicates and actions have at most arity two, including Blocksworld, Delivery, Logistics, and n-Puzzle (Jansen et al., 2025). Positive transitions are generated by a depth-first traversal of the reachable state space. A training iteration is considered successful only if the model achieves perfect accuracy on all instances across all actions for both the training and test sets. This criterion is intentionally strict because learned action models are to be used for long-horizon planning.

Table 2: Action model learning: % of runs (out of 10) where models achieved perfect accuracy.

	Precondition				Effect			
Model	**Blocksworld**	**Delivery**	**Logistics**	**n-Puzzle**	**Blocksworld**	**Delivery**	**Logistics**	**n-Puzzle**
NDLM (ours)	100%	100%	100%	100%	100%	100%	100%	100%
NLM	100%	100%	100%	100%	80.0%	100%	100%	50.0%
EdgeTransformer	0%	0%	0%	80%	100%	30%	50%	0%
PPGN	10%	0%	0%	0%	30%	0%	0%	0%
2GNN	50%	70%	0%	0%	0%	0%	0%	0%

As baseline we use NLMs (Dong et al. (2019)), EdgeTransformers (Bergen et al. (2021)), PPGNs (Maron et al. (2020)), and 2-GNNs (Morris et al. (2019)). All were given the same training and similar hyperparameter search setup, as well as the same training data. Further experimental details are provided in the Appendix B.4. NDLMs can fit the sampled training transitions and generalize to larger test instances, while the baseline models don't, with the exception of NLMs. The experiments are run on limited training data (see Appendix 14), which highlights NDLMs ability to learn and generalize from few examples. We further analyse the failure cases of the baseline models in Appendix B.4.1, with varying amounts of training data.

7.3 General Policy Learning

The general policy experiments use the formulation in Section 3.2. For each action schema, the model predicts the tuples of applicable explicit STRIPS+ action arguments that decrease the optimal distance to the goal. The experiments use the domains and training and testing instances from Ståhlberg et al. (2025), including Navigation, Blocksworld, Logistics, and Gripper that are mapped to STRIPS+ actions schemas with lower action arity (Jansen et al., 2025). Table 3 evaluates closed-loop rollouts obtained by repeatedly applying the highest-scoring applicable action above the decision threshold, until the goal is reached (=success), no action exceeds the decision threshold, or a cycle is detected (=failure). Table 4 shows that NDLM with the transitive closure 5.2.1 learns general policies in other domains. We use the same baselines as before.

Table 3: General policy learning: Number of instances (out of 10) with successful policies.

Approach	**Gripper**		**Logistics**	
	Train	**Test**	**Train**	**Test**
NDLM (ours)	13.0 ± 0.0 / 13	18.0 ± 0.0 / 18	24.0 ± 0.0 / 24	20.0 ± 0.0 / 20
NLM	13.0 ± 0.0 / 13	18.0 ± 0.0 / 18	24.0 ± 0.0 / 24	20.0 ± 0.0 / 20
EdgeTransformer	13.0 ± 0.0 / 13	14.3 ± 6.1 / 18	0.0 ± 0.0 / 0	0.0 ± 0.0 / 20
PPGN	4.7 ± 1.6 / 13	0.0 ± 0.0 / 18	0.1 ± 0.3 / 24	0.0 ± 0.0 / 20

As above, NDLM achieving strong accuracy on both training and test instances, while the baselines models struggle on both, except for NLMs that only fails in the n-puzzle. NDLMs show promising performance across domains like Navigation-xy and Logistics that require expressivity beyond 2-variable logic (Ståhlberg et al., 2025). EdgeTransformer and PPGN struggle to fit the training data, and hence fail to generalize effectively to test instances.

Comparison with NLMs NLMs performed very much like NDLMs in the planning experiments (with the exception of model learning in the n-puzzle), but require cubic space in the number of

Table 4: General policy learning: domains requiring transitive closure.

Approach	Blocksworld	Miconic	Navigation-xy
NDLM train	17.9 ± 0.3 / 18	10.0 ± 0.0 / 10	224.0 ± 0.0 / 224
NDLM test	16.0 ± 0.9 / 17	18.0 ± 0.0 / 20	96.0 ± 0.0 / 96

objects, versus quadratic space for NDLMs. On a 22-GB GPU, NLMs run out of memory at approximately 342 objects in the gripper domain, while NDLMs run *well* beyond 1,000. Memory plots appear in Appendix B.6.

8 INTERPRETABILITY

NDLMs are based on description logic but do not result in logical definitions of the learned predicates but in neural parametric functions. Yet the functions are not fully opaque, as we illustrate next. For example, the prediction of the add and delete effects of the STRIPS+ action $unstack(x)$ where x is the location of the block to unstack, results in expressions of this form after pruning irrelevant weights (Appendix F):

		Add	Delete
Unary	*clear*	$4.49 \cdot arg_0$	$3.81 \cdot \exists on.arg_0$
	holding	$3.78 \cdot \exists on.arg_0$	–
	handempty	–	$1.31 \cdot \top$
Binary	*on*	–	$2.30 \cdot on - 1.32 \cdot (handempty \triangleright \top_R) - 1.24 \cdot (\top \triangleright \top_R)^- + 2.17 \cdot (arg_0 \triangleright \top_R)^-$

Expressions of the form $p \leftarrow 0$, corresponding to $\bot$ or $\bot_R$, are omitted. The unary and binary predicates p represent add and delete changes, and $p(o)$ is true (resp. false) iff plugging o into the left-hand-side add (resp. delete) expression yields a strictly positive value. For example, $clear(o)$ is added by $unstack(x)$ iff $o = x$. In that case, the right-hand side add-$clear$ expression is positive, while for all other objects o' it is zero. The same reasoning applies to unary delete predicates and binary add/delete predicates. For example, the last line says that $on(a, b)$ is deleted by $unstack(x)$ iff $b = x$; then the expression for (a, b) evaluates to $2.30 - 1.32 - 1.24 + 2.17 > 0$, whereas for all other pairs it is negative. Here, $C \triangleright \top_R$ contains all pairs (o, o') with o satisfying C, and $(C \triangleright \top_R)^-$ contains all pairs with o' satisfying C. Across the Blocksworld models, pruning leaves 24 of 892 parameters (a 97.3% reduction; Appendix, Table 20).

9 CONCLUSIONS

We have introduced a new architecture for learning unary and binary predicates from examples. NDLMs are based on a rich description logic which has the expressive power of 3-variable first-order logic, FO^3, but unlike most other architectures that achieve this expressive power, including the closely related Neural Logic Machines, it uses quadratic rather than cubic space. NDLMs aim to combine the strengths of symbolic methods (generalization, precision, data efficiency, interpretability) with the strength of neural methods (scalability, noise-tolerance). The experimental results illustrate these properties in particular in settings where FO^3 contains perfect solutions. We have also shown that many key learning problems in planning, including general domain and policy learning can be mapped into predicate learning tasks that NDLMs can solve effectively. Unlike other neural architectures such as k-GNNs, however, NDLMs don't capture either FO^3 or FO^2 logics with counting quantifiers, and while being based on a fragment of first-order logic makes them suitable for learning predicates that generalize well, the performance on problems that require additional expressive power may be poor. There is indeed a well known tension between expressivity and generalization, and NDLMs capture one point in this spectrum that may be sufficiently practical.

AI USE STATEMENT

In this work, we used generative AI tools for formulate mathematical claims, assist in the writing of proofs, and design or provide feedback on research methodology or experiments, implement methods. We have not used generative AI tools for generate synthetic data sets, help develop theoretical models or conceptual frameworks, provide critical ingredients for proving mathematical claims, propose or refine hypotheses, assist with translation, clean and reformat dataset, support qualitative and thematic data analysis. Additionally, we used generative AI tools for create or modify scientific figures or images, draft parts of a research paper, transcribe recordings of research material, summarize or analyse existing literature, discover research topics or identify gaps, brainstorming, sourcing/searching for information, edit a research paper to improve readability, identify relevant literature. We have reviewed all AI-assisted work. All AI-generated code has been reviewed by the authors. AI-generated text has also been closely reviewed by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The supplementary material includes the complete codebase, including model implementations and experiment run scripts and datasets. The appendix documents the evaluation setups and concrete hyperparameter selections for the TUDataset graph-classification, OWL concept-learning, action-model-learning, and general-policy experiments. It also reports relevant cross-validation procedures, training and test instance statistics, and, for action-model learning, performance as the training-data size varies.

REFERENCES

Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook*. Cambridge University Press, 2007.

Leon Bergen, Timothy O'Donnell, and Dzmitry Bahdanau. Systematic generalization with edge transformers. *Advances in Neural Information Processing Systems*, 34:1390–1402, 2021.

Balder ten Cate and Victor Dalmau. Conjunctive queries: Unique characterizations and exact learnability. *ACM Transactions on Database Systems*, 47(4):1–41, 2022.

Andrew Cropper and Rolf Morel. Learning programs by learning from failures. *Machine Learning*, 110(4):801–856, 2021.

Maarten de Rijke. The logic of peirce algebras. *Journal of Logic, Language and Information*, 4(3): 227–250, 1995. doi: 10.1007/BF01049414.

Caglar Demir and Axel-Cyrille Ngonga Ngomo. Neuro-symbolic class expression learning. In *IJCAI*, volume 23, pp. 3624–3632, 2023.

Caglar Demir, Alkid Baci, N'Dah Jean Kouagou, Leonie Nora Sieger, Stefan Heindorf, Simon Bin, Lukas Blübaum, Alexander Bigerl, and Axel-Cyrille Ngonga Ngomo. Ontolearn—a framework for large-scale owl class expression learning in python. *Journal of Machine Learning Research*, 26(63):1–6, 2025.

Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural logic machines. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=B1xY-hRctX.

Dominik Drexler, Guillem Francès, and Jendrik Seipp. DLPlan, 2022. URL https://doi.org/10.5281/zenodo.5826139.

Guillem Frances, Blai Bonet, and Hector Geffner. Learning general planning policies from small examples without supervision. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 11801–11808, 2021.

Maurice Funk, Jean Christoph Jung, and Tom Voellmer. Sat-based bounded fitting for the description logic alc. In *International Semantic Web Conference*, pp. 42–60. Springer, 2025.

Jia He and Maggie X. Cheng. Orthogonal bases for equivariant graph learning with provable k-wl expressive power. *Journal of Machine Learning Research*, 26(29):1–35, 2025. URL http://jmlr.org.

Uwe Hustadt, Ralf A. Schmidt, and Lilia Georgieva. A survey of decidable first-order fragments and description logics. *Journal of Relational and Mathematical Logic*, 1:251–276, 2004.

Niklas Jansen, Jonas Gösgens, and Hector Geffner. Learning lifted action models from traces of incomplete actions and states. *arXiv preprint arXiv:2508.21449*, 2025.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL https://api.semanticscholar.org/CorpusID:6628106.

Jens Lehmann and Pascal Hitzler. Concept learning in description logics using refinement operators. *Machine Learning*, 78(1):203–250, 2010.

Jens Lehmann, Sören Auer, Lorenz Bühmann, and Sebastian Tramp. Class expression learning for ontology engineering. *Journal of Web Semantics*, 9(1):71–81, 2011.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks, 2020. URL https://arxiv.org/abs/1905.11136.

Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4602–4609, 2019. doi: 10.1609/aaai.v33i01.33014602.

Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond*, 2020. URL https://chrsmrrs.github.io/datasets/.

Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.

Stuart Russell. Artificial intelligence: A modern approach, 2010.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning general optimal policies with graph neural networks: Expressive power, transparency, and limits. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 32, pp. 629–637, 2022.

Simon Ståhlberg, Blai Bonet, and Hector Geffner. Learning more expressive general policies for classical planning domains. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 26697–26706, 2025. doi: 10.1609/aaai.v39i25.34872.

Balder ten Cate and Víctor Dalmau. The product homomorphism problem and applications. In *18th International Conference on Database Theory (ICDT 2015)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2015.

Aalok Thakkar, Aaditya Naik, Nathaniel Sands, Rajeev Alur, Mayur Naik, and Mukund Raghothaman. Example-guided synthesis of relational queries. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 1110–1125, 2021.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Pinar Yanardag and S.V.N. Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pp. 1365–1374, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336642. doi: 10.1145/2783258.2783417. URL https://doi.org/10.1145/2783258.2783417.

Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. *Advances in neural information processing systems*, 34:29476–29490, 2021.

CONTENTS

1 Introduction **1**

2 The Task: Predicate Learning **2**

3 Applications in Planning **2**

3.1 Action Model Learning 3

3.2 General Policy Learning 3

4 Related Work **3**

5 Architecture **4**

5.1 The Description Logic in NDLMs 4

5.2 Layers 4

5.2.1 Generating Intermediate Concepts and Roles 5

5.2.2 Combining Intermediate Predicates 6

5.3 Training NDLMs 6

5.4 Complexity 6

6 Expressivity **7**

7 Experiments **7**

7.1 TUDataset 7

7.2 Action Model Learning 8

7.3 General Policy Learning 8

8 Interpretability **9**

9 Conclusions **9**

A Extended Background **14**

A.1 STRIPS Semantics 14

A.2 Description-Logic Syntax and Semantics 14

B Experimental Details **15**

B.1 TUDataset 15

B.2 OWL Concept Learning 17

B.3 Planning Tasks: Common Setup 19

B.4 Action Model Learning 19

B.4.1 Dependence on Training Data Size 20

B.5 General Policy Learning 20

B.6 NDLM vs NLM, memory footprint 21

C Architecture: Transitive Closure **22**

D Theoretical Details **22**

D.1 On the relation to FO^3 22

D.2 Proof sketch of Theorem 6.1 22

D.3 Proof of Theorem 6.1 23

D.3.1 Lower Bound 23

D.3.2 Upper Bound 24

D.4 Complexity Details 25

E Related Work **26**

E.1 Symbolic approaches 26

E.2 Neural Architectures 26

F Interpretability **27**

A Extended Background

A.1 STRIPS Semantics

A STRIPS planning task is a tuple $\Pi = (O, P, A, s_0, g)$, where O is a finite set of objects, P is a finite set of predicate symbols, A is a set of action schemas, s_0 is the initial state, and g is the goal condition. Let $\mathsf{At}(P, O)$ denote the set of ground atoms formed from P and O. A state is a set $s \subseteq \mathsf{At}(P, O)$; an atom is true in s exactly when it belongs to s.

Each action schema $a(\bar{x}) \in A$ consists of parameters $\bar{x}$, a positive precondition set pre_a, an add-effect set add_a, and a delete-effect set del_a, all containing atoms over $\bar{x}$. A substitution θ maps the parameters to objects and yields the ground action $a\theta$. This action is applicable in s iff

$$\mathsf{pre}_a\theta \subseteq s. \tag{4}$$

Its deterministic successor state is

$$\gamma(s, a\theta) = (s \setminus \mathsf{del}_a\theta) \cup \mathsf{add}_a\theta. \tag{5}$$

The goal condition $g \subseteq \mathsf{At}(P, O)$ is satisfied by s iff $g \subseteq s$. In the main text, s is equivalently represented predicate-wise by the extensions p^s for $p \in P$.

A.2 Description-Logic Syntax and Semantics

Let $I = (\Delta, \cdot^I)$ be a finite interpretation with primitive concept names A and primitive role names P. We use the following language $\mathcal{D}$:

$$C ::= A \mid \top \mid \bot \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \exists R.C \mid \forall R.C,$$
$$R ::= P \mid \top_R \mid \bot_R \mid id \mid \neg R \mid R \sqcap S \mid R \sqcup S \mid R^- \mid R \circ S \mid C \triangleright R.$$

Here, C, D range over concepts and R, S over roles. The interpretation maps concepts to subsets of Δ and roles to subsets of Δ^2. Primitive symbols retain their given interpretation; the remaining

operators have the following semantics:

$$
\begin{aligned}
(\top)^I &= \Delta, \\
(\bot)^I &= \emptyset, \\
(id)^I &= \{(x,x) \mid x \in \Delta\}, \\
(\neg C)^I &= \Delta \setminus C^I, \\
(C \sqcap D)^I &= C^I \cap D^I, \\
(C \sqcup D)^I &= C^I \cup D^I, \\
(\exists R.C)^I &= \{x \in \Delta \mid \exists y \in C^I : (x,y) \in R^I\}, \\
(\forall R.C)^I &= \{x \in \Delta \mid \forall y \in \Delta : \ (x,y) \in R^I \Rightarrow y \in C^I\}, \\
(\top_R)^I &= \Delta^2, \\
(\bot_R)^I &= \emptyset, \\
(\neg R)^I &= \Delta^2 \setminus R^I, \\
(R \sqcap S)^I &= R^I \cap S^I, \\
(R \sqcup S)^I &= R^I \cup S^I, \\
(R^-)^I &= \{(y,x) \mid (x,y) \in R^I\}, \\
(R \circ S)^I &= \{(x,z) \in \Delta^2 \mid \\
&\qquad \exists y \in \Delta : \ (x,y) \in R^I \wedge (y,z) \in S^I\}, \\
(C \triangleright R)^I &= \{(x,y) \in R^I \mid x \in C^I\}.
\end{aligned}
$$

The last operator restricts a role at its source and is included because CE broadcasts a concept value over all outgoing role entries.

B Experimental Details

The general training procedure is described in Section 5.3. Here, we summarize the key details relevant to the experiments presented in this paper, these values were fixed for all experiments and did not change across different runs.

Parameter	Value	Description
learning rate	0.001	The step size used by the Adam optimizer during training.
weight decay	0.001	The weight decay (L2 regularization) applied during training.
batch size	4	The number of training examples used in one iteration.
epochs	500	The number of complete passes through the training dataset.
early stopping patience	10	The number of epochs with perfect accuracy on the whole training dataset before stopping early.

B.1 TUDataset

The same hyperparameter search space was used for NDLM and NLM. The 18 architectural configurations comprise all combinations of the layer counts, hidden widths, and activation functions shown in Table 6; learning rate and decay rate were varied over the listed values.

Table 5: Graph classification accuracy. Baseline results taken from He & Cheng (2025).

Method	MUTAG	PTC	PROTEINS	NCI1	NCI109	COLLAB	IMDB-B	IMDB-M
WEGL	88.3	67.5	76.5	–	–	78.6	72.0	48.5
DGK	87.4	60.0	75.6	80.3	80.3	73.1	66.9	44.5
DGCNN	85.8	58.6	75.5	74.4	–	73.7	70.0	47.8
2-IGN	84.61	59.0	75.19	74.3	72.8	78.3	72.0	48.7
PPGN	90.6	66.2	77.2	83.2	82.2	83.1	73.0	50.4
GIN	89.4	64.6	76.2	82.7	–	80.2	75.1	52.3
1-2-3 GNN	86.1	60.9	75.5	76.2	–	–	74.2	49.5
GFN	90.84	–	76.46	82.7	–	80.4	73.3	51.2
DropGIN	90.4	66.3	76.3	–	–	–	75.7	51.4
InfoGraph	89.01	61.65	–	–	–	–	73.0	49.7
BC + Capsules	88.9	69.0	74.1	65.9	58.0	–	–	–
hGNet	90.0	65.02	78.65	–	–	77.5	–	49.1
CCN	91.64	70.62	–	76.27	75.54	–	–	–
CIN	92.7	68.2	77.0	83.6	**84.0**	–	75.6	52.7
GSN	92.2	68.2	76.6	83.5	–	**85.5**	**77.8**	54.3
SIN	–	–	76.4	82.7	–	–	75.6	52.4
Order-2 GNN-b	92.0	**72.4**	73.9	80.0	81.0	78.5	72.4	51.0
Order-3 GNN-a	94.4	72.1	77.8	**85.0**	83.4	80.7	74.9	52.5
NLM	**97.22**	67.65	76.58	80.29	80.26	71.80	74.50	**54.67**
NDLM (ours)	88.89	66.18	**80.63**	81.75	80.44	79.30	72.67	**54.67**

Table 6: Hyperparameter values used in the TUDataset search for both NDLM and NLM.

Hyperparameter	Values
Layers	$\{4, 6, 8\}$
Hidden width	$\{5, 10, 15, 20\}$
Activation	$\{\texttt{identity}, \texttt{sigmoid}\}$
Learning rate	$\{0.00005, 0.0001, 0.0005, 0.001\}$
Decay rate	$\{0.5, 1.0\}$

Hyperparameters were selected using fold 0 as a train-validation split. The selected NLM configurations used in the 10-fold cross-validation runs are shown in Table 7; the corresponding NDLM configurations and reported cross-validation results are shown in Table 8.

Table 7: Selected NLM hyperparameters for the TUDataset cross-validation runs.

Dataset	Layers	Width	Activation	Decay rate
COLLAB	4	10	identity	0.5
IMDBBINARY	4	10	identity	1.0
IMDBMULTI	4	5	identity	1.0
MUTAG	4	5	identity	1.0
NCI1	4	10	identity	0.5
NCI109	4	15	identity	0.5
PROTEINS	4	10	identity	1.0
PTC	4	5	identity	1.0

All NDLM configurations below were evaluated over 10 folds.

Table 8: Selected NDLM hyperparameters for TUDataset. IMDB-B and IMDB-M denote IMDB-BINARY and IMDBMULTI, respectively.

Dataset	Layers	Width	Activation	Decay
COLLAB	4	10	identity	0.5
IMDB-B	6	15	identity	0.5
IMDB-M	4	20	identity	0.75
MUTAG	4	20	identity	1.0
NCI1	4	20	identity	0.75
NCI109	4	20	identity	0.75
PROTEINS	4	10	sigmoid	0.5
PTC	4	10	identity	1.0

B.2 OWL Concept Learning

In addition to the experiments in the main text, we compare NDLM against several state-of-the-art symbolic concept learners.

We evaluate NDLM on relational OWL concept-learning benchmarks from OntoLearn Demir et al. (2025). We consider the BioPAX, Carcinogenesis, Carcinogenesis*, Family, Mutagenesis, Mutagenesis*, Lymphography, and Nctrer domains. The domains marked with * use an extended set of learning problems in which known description-logic concepts are generated randomly and examples are sampled accordingly. Additional experimental details are provided in Appendix B.2.

For each learning problem the training dataset is composed of individual training datapoints, each of which is given by the local neighborhood of a queried individual. The model predicts a unary membership label over this neighborhood, but the loss is evaluated only at the queried individual; predictions for all other individuals are ignored. Local-neighborhood encoding is used only for NDLM, whereas the other concept learners are provided with the full knowledge graph.

Tables 9 and 10 report macro-F1 scores against state-of-the-art symbolic, neuro-symbolic, evolutionary, and tree-based baselines. F1 scores are computed for each target concept (learning problem) and then averaged to obtain the macro-F1. All experiments are conducted using 10-fold cross-validation, where the same fold split was used across all learners. Evaluated hyperparameter selections are reported in Appendix B.2.

Each learner, including NDLM, was restricted to 60 seconds per fold. This time limit excluded loading the knowledge graph and constructing local neighborhoods.

Table 9: Macro-F1 (**mean** $\pm$ **std**) on domains with known ground-truth formulas. Best per-domain means are bold.

Approach	Paradigm	Known ground truth formula; multi-problem			
		Biopax	Carcinogenesis*	Family	Mutagenesis*
NDLM	Neural	99.38 ± 1.97	99.00 ± 1.54	93.02 ± 4.77	98.03 ± 4.69
ALCSAT	SAT-based	99.51 ± 1.86	99.47 ± 1.23	99.05 ± 1.63	$\mathbf{98.93 \pm 2.32}$
CELOE	Search-based	99.26 ± 2.02	99.10 ± 1.49	93.92 ± 7.98	98.50 ± 2.39
Drill	Neuro-symbolic	$\mathbf{99.69 \pm 1.75}$	$\mathbf{100.00 \pm 0.00}$	87.96 ± 8.43	95.83 ± 11.36
EvoLearner	Evolutionary	99.39 ± 1.97	99.67 ± 1.30	$\mathbf{99.16 \pm 1.08}$	95.23 ± 11.33
OCEL	Search-based	99.06 ± 2.22	99.10 ± 1.49	90.06 ± 14.14	96.60 ± 8.99
TDL	Tree-based	94.98 ± 4.99	98.63 ± 3.70	92.38 ± 5.27	96.00 ± 7.86

Table 10: Macro-F1 on domains with unknown target formulas; best per-domain means are bold.

Approach	Paradigm	Unknown formula; single-problem			
		Carcinogenesis	Lymphography	Mutagenesis	Nctrer
NDLM	Neural	70.39	83.32	75.43	73.81
ALCSAT	SAT-based	70.39	70.74	70.39	73.81
CELOE	Search-based	71.94	**86.86**	89.91	98.80
Drill	Neuro-symbolic	70.39	**86.86**	76.15	75.41
EvoLearner	Evolutionary	**72.63**	83.99	**95.57**	**100.00**
OCEL	Search-based	70.39	**86.86**	89.91	98.80
TDL	Tree-based	64.95	81.02	86.46	**100.00**

NDLM achieves results comparable to several symbolic and neuro-symbolic baselines, particularly on the known-formula domains. These results are obtained by training NDLM from scratch within the 60-second time limit per fold.

Table 11 describes general statistics of the domains used in the OWL concept learning experiments.

Table 11: Statistics for the OWL concept-learning domains. Asterisks denote drill-derived learning-problem sets

Domain	#Objects	#Concepts	#Roles	#Problems	Avg. Pos.	Avg. Neg.
BioPAX	447	21	21	96	10.34	10.34
Carcinogenesis*	22541	114	8	100	10.00	10.00
Family	230	18	5	7	32.14	32.14
Mutagenesis*	14250	61	7	100	10.00	10.00
Carcinogenesis	22541	114	8	1	55.00	46.00
Mutagenesis	14250	61	7	1	125.00	105.00
Lymphography	201	50	0	1	81.00	67.00
Nctrer	10317	16	10	1	131.00	93.00

The final results are obtained by evaluating NDLM with fixed hyperparameters per domain. The best scoring hyperparameters per domain are shown in Table 12.

Table 12: Best NDLM configuration and neighborhood selected for each domain. Width denotes the number of hidden concepts and hidden roles. Due to 60 second timelimit per fold, small neighborhoods performed best, as convergence was faster.

Domain	Layers	Width	Activation	Neighborhood
Biopax	3	10	identity	1
Carcinogenesis	5	5	sigmoid	1
Carcinogenesis*	3	10	identity	1
Family	3	10	identity	6
Lymphography	3	5	sigmoid	1
Mutagenesis	3	5	identity	1
Mutagenesis*	3	10	sigmoid	1
Nctrer	3	10	identity	1

The hyperparameter search evaluated all combinations of the following values:

Table 13: Hyperparameter values evaluated for OWL concept learning.

Hyperparameter	Values
Width	$\{5, 10, 15\}$
Depth	$\{3, 5\}$
Activation	$\{\texttt{identity}, \texttt{sigmoid}\}$
Neighborhood	$\{1, 2, 4, 6, 8\}$

B.3 Planning Tasks: Common Setup

Table 14 summarizes the number of supervised examples and instances per task and domain.

Table 14: Average number of supervised examples per action and corresponding numbers of training and evaluation instances for each domain. For general-policy learning, "–" denotes no separate supervised test split.

Task	Domain	Avg. train	Avg. test	Train instances	Test instances
Action model (effect)	Blocksworld	162.8	345.8	2	32
Action model (effect)	Delivery	267.7	42.3	2	2
Action model (effect)	Logistics	960.8	20.3	7	2
Action model (effect)	n-Puzzle	93.3	23.5	1	2
Action model (precondition)	Blocksworld	650.0	4154.0	2	32
Action model (precondition)	Delivery	822.0	326.0	2	2
Action model (precondition)	Logistics	1622.0	32.0	7	2
Action model (precondition)	n-Puzzle	360.0	474.0	1	2
General policy	Blocksworld	1493.5	–	18	17
General policy	Gripper	1534.3	–	13	18
General policy	Logistics	2902.2	–	24	20
General policy	Miconic	873.0	–	10	20
General policy	Navigation-xy	2034.8	–	224	96

B.4 Action Model Learning

Preconditions and effects are learned as separate tasks, without an extensive hyperparameter search. All models use batch size 4 and are trained for at most 500 epochs on effects and 1000 epochs on preconditions; 2GNN stops early once training has converged. For NDLM, the training and test sets are generated from 200 and 30 expanded states for effects and from 600 and 300 expanded states for preconditions. Negative precondition examples are obtained by sampling 10 random ground actions per state. Table 15 summarizes the selected architecture settings, and Table 16 the object-count ranges of the instances.

Table 15: Selected action-model configurations for effect and precondition learning.

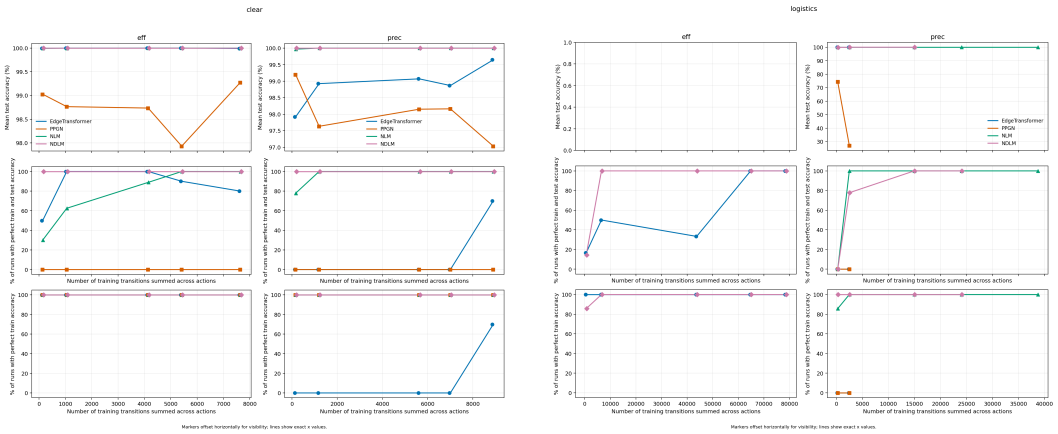
Model	Effects	Preconditions
NDLM	3 layers; sigmoid activation	4 layers; width 10; identity activation
NLM	6 layers; width 5	6 layers; width 5
2GNN	width 128	3 layers; hidden size 64; identity activation
EdgeTransformer	4 layers; width 10; 2 heads	hidden size 64; 3 layers; 4 heads; activation not set
PPGN	width 400; learning rate 10^{-4}	width 400; learning rate 10^{-4}

Table 16: Object-count ranges of the training and test instances used for action model learning.

Domain	Train objects	Test objects
Logistics	12–20	35–45
Delivery	16–22	32–55
Blocksworld	5–10	3–14
n-Puzzle	11	11–15

B.4.1 Dependence on Training Data Size

Figure 2 illustrates how NDLM performance depends on the number of training examples. For each domain, we report accuracy, test success, and training success over 10 runs.



(a) Blocksworld (b) Logistics

(c) Delivery (d) n-Puzzle

Figure 2: Action-model learning performance as the number of training examples varies. Panels show Blocksworld, Logistics, Delivery, and n-Puzzle; each reports accuracy, test success, and training success over 10 runs.

B.5 General Policy Learning

Table 18 summarizes the number and object-count ranges of the training and test instances used for general policy learning.

Table 17: Hyperparameter settings for general policy learning by domain.

Domain	Depth	Width	Activation function	Transitive Closure
blocks	4	10	identity	True
gripper	3	10	identity	False
logistics	6	10	identity	False
miconic	4	10	identity	True
navigation-xy	3	5	identity	True

Table 18: Train and test ranges give the minimum and maximum number of objects per instance; # Train and # Test give the numbers of instances.

Domain	Train range	Test range	# Train	# Test
blocks	4–9	10–17	18	17
grid	13–45	25–96	35	20
gripper	6–18	20–54	13	18
logistics	12–24	69–88	24	20
miconic	3–30	33–90	10	20
navigation-xy	4–11	8–38	224	96

B.6 NDLM VS NLM, MEMORY FOOTPRINT

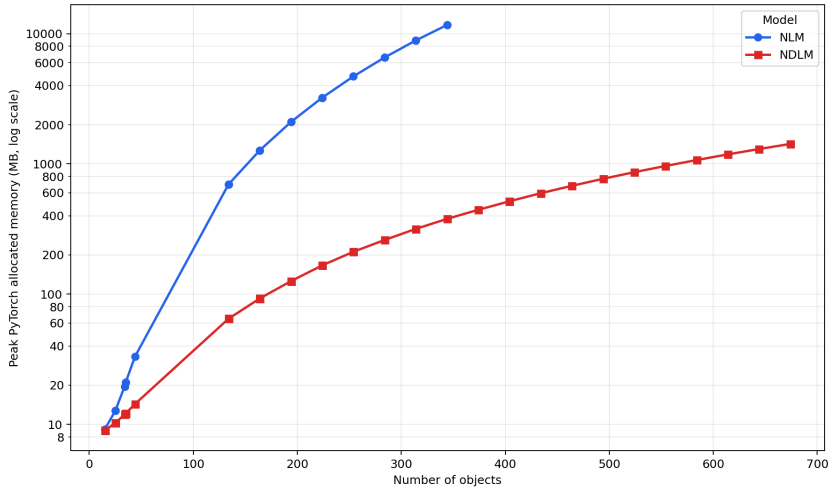


Figure 3: Forward-pass latency and peak GPU memory usage of NDLMs and NLMs. NDLM: 4 layers, width 10, NLM: 4 layers, width 5.

NLMs (see Section E.2) are closely related to NDLMs. However, NDLMs are capable of achieving the FO^3 expressivity while using only quadratic memory for binary-predicate inputs, unlike NLMs which require cubic memory. We demonstrate experimentally that the memory demand of NLMs becomes a real bottleneck as the number of objects increases. This is shown in Figure 3.

We compare the memory footprint between NDLM and NLM of the learned models for general policy learning in the gripper domain. Both results are run on an NVIDIA A10 GPU with 24 GB VRAM, we use float32 precision and, evaluate instances with increasing number of objects, we report peak allocated CUDA memory. Figure 3 clearly illustrates the substantial memory savings of NDLMs compared to NLMs as the number of objects increases.

C ARCHITECTURE: TRANSITIVE CLOSURE

For an input of n objects, the transitive closure can be computed iteratively as

$$TC^s(R) = \max_{i \in [n-1]} TC^s_i(R)$$

with $TC^s_0(R)_{i,j} = I_{i,j}$ and $TC^s_i(R) = \max_{p \in [n]}\{\min(TC^s_i(R)_{i,p}, R_{p,j})\}$ for $i \in [k]$. TC^s_i encodes the reachability for paths of fixed length i.

D THEORETICAL DETAILS

Throughout this appendix, we write CRA for Role Quantification (RQ), RRA for Role Composition (RC), and RA for Role Reduction (RR) from Section 5; CE and RT keep their names, and a subscript, as in $\mathrm{CRA}_{\max}$, selects the channels computed with the corresponding aggregator.

D.1 ON THE RELATION TO FO^3

Let $\mathrm{FO}^3_{\leq 2}$ denote the three-variable fragment of first-order logic over predicates of arity at most two. The description logic $\mathcal{D}$ defined in Section 5.1 and Appendix A.2 has the expressive power of $\mathrm{FO}^3_{\leq 2}$.

Hustadt et al. Hustadt et al. (2004) establish that

$$\mathrm{ALC}(\neg, \sqcap, ^{-1}, \circ, id) \equiv_{\mathrm{expr}} \mathsf{PL},$$

where PL denotes Peirce logic. De Rijke de Rijke (1995) establishes the expressive completeness of the Peirce modal language with respect to the three-variable fragment of first-order logic. Consequently,

$$\mathsf{PL} \equiv_{\mathrm{expr}} \mathrm{FO}^3_{\leq 2}.$$

Our language $\mathcal{D}$ contains the corresponding relational constructors: complement, intersection, inverse, composition, and identity—together with their Boolean extensions and domain restriction. These additional constructors are expressible by the basic ones and do not increase the expressive power. Hence,

$$\boxed{\mathcal{D} \equiv \mathrm{expr}\mathrm{FO}^3_{\leq 2}}$$

D.2 PROOF SKETCH OF THEOREM 6.1

Proof sketch. **Lower bound.** We construct a network by structural induction on F. Primitive concepts and roles are copied from the input. Pointwise FFNs implement Boolean operators, RT implements role inversion, and CE followed by a pointwise role FFN implements source restriction. The max channel of RQ implements existential restriction; universal restriction follows from $\forall R.C \equiv \neg\exists R.\neg C$ and pointwise negation. The max channel of RC implements role composition. Combining the subnetworks for the immediate subformulas in separate intermediate channels yields a network for F.

Upper bound. For each layer, partition objects and object pairs into types according to the information available to the layer: their current type and the sets of types occurring in the inputs to RR, RQ, CE, RT, and RC. Equal types receive equal outputs because the aggregations ignore multiplicities and the FFNs are pointwise. By induction, every type is definable by a formula in $\mathcal{D}$; moreover, for fixed depth and a finite input signature, only finitely many such types can occur across all finite interpretations. The output is constant on each final type, so the finite disjunction of the formulas defining precisely the types whose output is at least τ yields $F_{N,\tau} \in \mathcal{D}$. Appendix D.3 gives the full construction.

D.3 PROOF OF THEOREM 6.1

We prove the lower and upper bounds in Theorem 6.1. Throughout, the signature consists of finite sets N_C of primitive concept names and N_R of primitive role names. We write $\mathbf{1}_X$ for the Boolean indicator of a set X.

Again we assume that the pointwise CONCEPTFFN and ROLEFFN modules can realize the required Boolean functions exactly on Boolean inputs. This assumption isolates the structural expressivity of the architecture from the choice of activation function. In the experiments both sigmoid and identity activations were utilized. For sigmoid, boolean operations can be approximated arbitrarily well by choosing appropriate weights and biases, with identity activations the network can in principle also learn them, although the implementation needs also the first stage to introduce some nonlinearity. We skip the details and for the analysis assume exact realization of Boolean functions by the FFNs.

D.3.1 LOWER BOUND

We prove both claims in part 1 of Theorem 6.1 simultaneously by structural induction on the formula F. The induction invariant is that the constructed NDLM produces $\mathbf{1}_{F^I}$ for every finite interpretation I.

Table 19 gives the construction for every operator in $\mathcal{D}$. An FFN in the table denotes a pointwise FFN parameterized with the indicated Boolean function. The operands of each operator are assumed to be available as intermediate predicate channels by the induction hypothesis.

Table 19: Translation of description-logic operators into NDLM modules.

Formula type	$\mathcal{D}$ operator	NDLM construction
Concept	primitive A	input concept channel
Concept	$\top, \bot$	CONCEPTFFN with constant output
Concept	$\neg C, C \sqcap D, C \sqcup D$	pointwise CONCEPTFFN
Concept	$\exists R.C$	max channel of $\mathrm{CRA}(R, C)$
Concept	$\forall R.C$	$\neg\, \mathrm{CRA}_{\max}(R, \neg C)$
Role	primitive P	input role channel
Role	*id*	input role channel
Role	$\top_R, \bot_R$	ROLEFFN with constant output
Role	$\neg R, R \sqcap S, R \sqcup S$	pointwise ROLEFFN
Role	R^-	$\mathrm{RT}(R)$
Role	$R \circ S$	max channel of $\mathrm{RRA}(R, S)$
Role	$C \triangleright R$	$\mathrm{ROLEFFN}(\mathrm{CE}(C), R)$

For completeness, we verify the non-pointwise cases separately.

Existential restriction $\exists R.C$. The max channel of CRA satisfies

$$\max_{y \in \Delta} \mathbf{1}_{R^I}(x, y)\, \mathbf{1}_{C^I}(y) = 1 \quad \Longleftrightarrow \quad x \in (\exists R.C)^I.$$

Universal restriction $\forall R.C$. This operator follows from the standard duality $\forall R.C \equiv \neg\exists R.\neg C$, implemented by two pointwise negations around the existential construction.

Role composition $R \circ S$. The max channel of RRA satisfies

$$\max_{y \in \Delta} \mathbf{1}_{R^I}(x, y)\, \mathbf{1}_{S^I}(y, z) = 1 \quad \Longleftrightarrow \quad (x, z) \in (R \circ S)^I.$$

Source restriction $C \triangleright R$. The CE module produces a role tensor whose (x, y)-entry is $\mathbf{1}_{C^I}(x)$. A pointwise role FFN combines this channel with $\mathbf{1}_{R^I}(x, y)$ and therefore realizes source restriction.

Thus every constructor of $\mathcal{D}$ is implemented by a bounded number of layers. Independent subformulas can occupy separate intermediate channels, so stacking the constructions along the syntax tree of F proves the lower bound.

D.3.2 Upper Bound

We first establish the upper bound on a fixed interpretation and then lift it to all finite interpretations. Let N be an NDLM of depth k, and let C_i and R_i be the concept and role tensors after layer i, where C_0 and R_0 encode the primitive predicates.

Proposition D.1 (Fixed-interpretation distinguishability)**.** For every finite interpretation $I = (\Delta, \cdot^I)$ and layer $i \in \{0, \dots, k\}$, there are equivalence relations $\equiv_i^C$ on Δ and $\equiv_i^R$ on Δ^2 such that:

1. if $x \equiv_i^C y$, then $C_i(:, x) = C_i(:, y)$;
2. if $(x, y) \equiv_i^R (u, v)$, then $R_i(:, x, y) = R_i(:, u, v)$;
3. every equivalence class of $\equiv_i^C$ and $\equiv_i^R$ is definable by a concept or role formula in $\mathcal{D}$, respectively.

Proof. At layer zero, define

$$x \equiv_0^C y \iff \bigwedge_{A \in \mathsf{N}_C} \big(x \in A^I \iff y \in A^I\big),$$
$$(x, y) \equiv_0^R (u, v) \iff \bigwedge_{P \in \mathsf{N}_R \cup \{id\}} \big((x, y) \in P^I \iff (u, v) \in P^I\big).$$

The first two claims hold by construction. Their classes are defined by conjoining every primitive predicate that holds and negating every primitive predicate that does not hold.

Assume the claims hold at layer i. Let $[x]_i^C$ and $[(x, y)]_i^R$ denote the current equivalence classes. Define auxiliary equivalences by the following signatures:

$$\begin{aligned}
\mathrm{agg}_i(x) &= \{[(x, z)]_i^R \mid z \in \Delta\},\\
\mathrm{cra}_i(x) &= \{([z]_i^C, [(x, z)]_i^R) \mid z \in \Delta\},\\
\mathrm{ext}_i(x, y) &= [x]_i^C,\\
\mathrm{tr}_i(x, y) &= [(y, x)]_i^R,\\
\mathrm{rra}_i(x, y) &= \{([(x, z)]_i^R, [(z, y)]_i^R) \mid z \in \Delta\}.
\end{aligned}$$

Two entries are auxiliary-equivalent when their corresponding signatures are equal. The NDLM aggregations in RA, CRA, and RRA depend only on these signatures: min and max ignore multiplicities and preserve equality whenever the set of input-value types is the same. Likewise, CE and RT depend only on ext_i and tr_i. Since pointwise FFNs map equal input vectors to equal output vectors, define

$$\begin{aligned}
x \equiv_{i+1}^C y &\iff x \equiv_i^C y \ \wedge\ \mathrm{agg}_i(x) = \mathrm{agg}_i(y) \ \wedge\ \mathrm{cra}_i(x) = \mathrm{cra}_i(y),\\
(x, y) \equiv_{i+1}^R (u, v) &\iff (x, y) \equiv_i^R (u, v) \ \wedge\ \mathrm{ext}_i(x, y) = \mathrm{ext}_i(u, v)\\
&\qquad \wedge\ \mathrm{tr}_i(x, y) = \mathrm{tr}_i(u, v) \ \wedge\ \mathrm{rra}_i(x, y) = \mathrm{rra}_i(u, v).
\end{aligned}$$

This proves claims 1 and 2 at layer $i + 1$.

It remains to show definability. By the induction hypothesis, let χ_U and ρ_B denote formulas defining a concept class U and role class B at layer i. If $\mathcal{B}_i$ and $\mathcal{U}_i$ are the finite sets of role and concept classes, then the auxiliary signatures above are defined as follows:

$$\begin{aligned}
\phi_{\mathrm{agg}_i(x)} &= \bigwedge_{B \in \mathrm{agg}_i(x)} \exists \rho_B.\top \ \sqcap \bigwedge_{B \in \mathcal{B}_i \setminus \mathrm{agg}_i(x)} \neg \exists \rho_B.\top,\\
\phi_{\mathrm{cra}_i(x)} &= \bigwedge_{(U,B) \in \mathrm{cra}_i(x)} \exists \rho_B.\chi_U \ \sqcap \bigwedge_{(U,B) \in (\mathcal{U}_i \times \mathcal{B}_i) \setminus \mathrm{cra}_i(x)} \neg \exists \rho_B.\chi_U,\\
\rho_{\mathrm{ext}_i(x,y)} &= \chi_{[x]_i^C} \rhd \top_R,\\
\rho_{\mathrm{tr}_i(x,y)} &= \big(\rho_{[(y,x)]_i^R}\big)^-,\\
\rho_{\mathrm{rra}_i(x,y)} &= \bigwedge_{(B,D) \in \mathrm{rra}_i(x,y)} (\rho_B \circ \rho_D) \ \sqcap \bigwedge_{(B,D) \in (\mathcal{B}_i \times \mathcal{B}_i) \setminus \mathrm{rra}_i(x,y)} \neg (\rho_B \circ \rho_D).
\end{aligned}$$

The first two formulas assert exactly which role classes, or pairs of concept and role classes, occur at an outgoing neighbor. The latter three formulas define the source extension, transpose, and composition signatures. Conjoining the formulas for the current class and all relevant auxiliary signatures defines each class of $\equiv^C_{i+1}$ and $\equiv^R_{i+1}$. This completes the induction. □

Proposition D.2 (Finitely many layer types)**.** For every layer $i \in \{0, \dots, k\}$, there are finite, interpretation-independent sets $\mathcal{U}_i$ and $\mathcal{B}_i$ of concept and role types. Every type in $\mathcal{U}_i$ and $\mathcal{B}_i$ is defined uniformly over all finite interpretations by a concept or role formula in $\mathcal{D}$, respectively. Moreover, entries with the same type, including entries from different interpretations, have the same layer-i vector.

Proof. At layer zero, let the concept and role types be the truth vectors of the primitive concept and role names. Thus, $|\mathcal{U}_0| \le 2^{|\mathsf{N}_C|}$ and $|\mathcal{B}_0| \le 2^{|\mathsf{N}_R|}$, and each type is uniformly defined by the conjunction of the corresponding primitive predicates and their negations.

Suppose that $\mathcal{U}_i$ and $\mathcal{B}_i$ have been constructed. For an object x and an object pair (x, y) in an interpretation I, define their next types by

$$\mathrm{tp}^C_{i+1}(x) = \Big(\mathrm{tp}^C_i(x),\ \{\mathrm{tp}^R_i(x,z) \mid z \in \Delta\},\ \{(\mathrm{tp}^C_i(z), \mathrm{tp}^R_i(x,z)) \mid z \in \Delta\}\Big),$$
$$\mathrm{tp}^R_{i+1}(x,y) = \Big(\mathrm{tp}^R_i(x,y), \mathrm{tp}^C_i(x), \mathrm{tp}^R_i(y,x),\ \{(\mathrm{tp}^R_i(x,z), \mathrm{tp}^R_i(z,y)) \mid z \in \Delta\}\Big).$$

Let $\mathcal{U}_{i+1}$ and $\mathcal{B}_{i+1}$ contain all tuples of these forms over $\mathcal{U}_i$ and $\mathcal{B}_i$. They are finite; in particular,

$$|\mathcal{U}_{i+1}| \le |\mathcal{U}_i|\, 2^{|\mathcal{B}_i|}\, 2^{|\mathcal{U}_i||\mathcal{B}_i|},$$
$$|\mathcal{B}_{i+1}| \le |\mathcal{B}_i|\, |\mathcal{U}_i|\, |\mathcal{B}_i|\, 2^{|\mathcal{B}_i|^2}.$$

For uniform definability, let χ_U and ρ_B define types $U \in \mathcal{U}_i$ and $B \in \mathcal{B}_i$. The positive and negative conjunctions in the proof of Proposition D.1 define, uniformly, each of the two concept signatures and the three role signatures occurring above. Conjoining these signature formulas with χ_U or ρ_B defines the corresponding tuple type. Finally, equal tuple types have equal inputs to every aggregation, CE, and RT; the pointwise FFNs therefore produce equal layer-$i+1$ vectors. This establishes the induction. □

To prove part 2 of Theorem 6.1, apply Proposition D.2 to the final layer. Every final type has a formula in $\mathcal{D}$ that defines it uniformly over all finite interpretations, and N assigns one fixed output value to that type. There are finitely many final types, so the disjunction of the formulas for exactly those types with output at least τ is a finite formula $F_{N,\tau}$. By construction, this formula agrees with the thresholded output of N on every finite interpretation, proving the upper bound.

D.4 Complexity Details

Consider a single NDLM layer with c input concepts, r input roles, c' output concepts, and r' output roles over n objects. The intermediate concept tensor contains the input concepts, $2rc$ CRA channels, and $2r$ RA channels, for a total of

$$d_C = c + 2r(c+1)$$

channels. The factors of two in CRA, RA, and RRA account for the separate minimum and maximum aggregations. The intermediate role tensor contains the input roles, $2r^2$ RRA channels, c CE channels, and r RT channels, for a total of

$$d_R = 2r + 2r^2 + c.$$

channels. Hence the intermediate activations use $O(d_C n + d_R n^2)$ memory, in addition to the $c'n + r'n^2$ output entries.

The RRA operation computes two max-min products for every ordered pair of input roles and costs $O(r^2 n^3)$ time; all remaining operations cost at most $O((d_C + d_R)n^2)$. Thus, for fixed depth and width, the complete NDLM has $O(n^3)$ forward-pass time and $O(n^2)$ activation memory.

The ConceptFFN has a weight matrix with $c'd_C$ entries and a bias vector with c' entries, hence $c'(d_C + 1)$ learnable parameters. Similarly, the RoleFFN has $r'd_R$ weights and r' biases, hence $r'(d_R + 1)$ parameters. The layer therefore has

$$c'\big(c + 2r(c+1) + 1\big) + r'\big(2r + 2r^2 + c + 1\big)$$

learnable parameters.

E Related Work

In the following we will review first symbolic approaches and the neural approaches to predicate learning.

E.1 Symbolic approaches

Concept learning. Concept learning approaches in description logics (DL) take as input a set of unary and binary predicates, also called concepts and roles, together with examples of a unary target predicate (Lehmann & Hitzler, 2010). Many learning approaches are refinement based, such as CELOE Lehmann et al. (2011), and search for a fitting concept top-down, starting at the most general concepts. For CELOE the search is guided by a heuristic, while Drill (Demir & Ngomo, 2023) learns the heuristic with deep reinforcement learning. Bounded fitting in comparison searches for a fitting concept in increasing size using a SAT solver for each size. This obtains a concept of minimal size which yields PAC guarantees on generalization.

Inductive logic programming. Inductive logic programming (ILP) (Muggleton & De Raedt, 1994) solves a predicate learning problem learning a logic program. The predicates are given as background knowledge and the target predicate as grounded positive and negative examples, both of which are, unlike in concept learning, not restricted in arity. Popper Cropper & Morel (2021) searches for a fitting program in increasing size and thus, like bounded fitting, returns a smallest program. There are also differentiable systems like δILP that learn a program using gradient descent but the resulting program is still symbolic and based on a set of template programs.

Conjunctive query fitting. Conjunctive query fitting (ten Cate & Dalmau, 2015) asks for a conjunctive query that returns all positive and no negative example tuples, and thus directly handles targets of higher arity. A fitting query exists if and only if the direct product of the positive examples has no homomorphism to any negative example; in that case, the canonical query of this product is the most specific fitting query.

E.2 Neural Architectures

NLMs. Neural logic machines (NLMs) (Dong et al., 2019) are a neurosymbolic architecture for logical reasoning on relational data. Predicates are represented as tensors whose order corresponds to the arity of the predicate. The architecture operates layerwise, and each layer transforms the tensors by fixed operations, namely permutation of the object dimensions, reduction to lower arity by quantification, and expansion to higher arity by broadcasting. The resulting predicates are then combined by an MLP that is applied tuple-wise, so the computation is equivariant under permutations of the objects. To obtain expressivity of FO^3, tensors of order at least three are required, resulting in memory cubic in the number of objects.

GNNs. Message-passing graph neural networks (GNNs) operate on graphs, where node features are updated by message passing of connected nodes. Graph and node classification are a special case of predicate learning with nullary and unary target predicates, respectively. The ability of GNNs to distinguish graphs is bounded by the 1-Weisfeiler–Leman test and therefore by C^2, which means that properties requiring three variable formulas cannot be learned Morris et al. (2019); Xu et al. (2019). To overcome this limitation, 3-GNNs, which do message passing between 3-sets of nodes, can be used, with the effect that the architecture is more expressive than C^2 but it has cubic space complexity. PPGN Maron et al. (2020) instead keeps a state for each node pair and updates the node-pair features by matrix multiplication. By doing this, it achieves C^3 expressivity while having

quadratic space and cubic time complexity. This is similar to NDLMs, with the difference that in NDLM the summation in the matrix multiplication is replaced by $\min$ and $\max$. Since $\min$ and $\max$, unlike sums, do ignore multiplicities, NDLMs cannot count and express properties that there are at least four successor. That means, their expressivity is incomparable to GNNs as bounded by C^2, and NDLMs can learn three variable formulas.

Edge Transformers. Edge Transformers (Bergen et al., 2021) take as input a fully connected graph and maintain a state for each ordered pair of objects. These states are updated using *triangular attention* which computes, for each triplet (i, k, j) a value, based on the states of the edges (i, k) and (k, j) and an attention score. The state of each edge (i, j) is then updated based on the values of the triplets (i, k, j) for all objects k. This aggregation mirrors relational composition, where the intermediate object k takes the role of the existentially quantified variable. This gives the approach the ability to learn three-variable formulas. Although only n^2 edge states are maintained, for each triplet (i, k, j) a value is stored, so both runtime and memory are cubic in the number of objects.

F INTERPRETABILITY

NDLMs represent learned predicates as neural parametric functions rather than explicit logical definitions. Their learned functions are not entirely opaque, however: pruning and translating the network operations can expose compact, semi-symbolic descriptions of its predictions. This is possible in part because of the correspondence between NDLMs and formulas in the description logic $\mathcal{D}$ established in Section 6.

To obtain these descriptions, we greedily consider the trainable parameters Θ in ascending order of absolute magnitude, setting a parameter to zero whenever doing so leaves the discrete predictions on the training set unchanged. We then replace the relational operations in the first stage of each layer with their logical counterparts and express identity-activation FFNs as linear combinations. In the Blocksworld effect models, pruning leaves 24 of 892 parameters while retaining the learned predictions on the training set. The per-action counts are shown in Table 20.

Table 20: Remaining and total trainable parameters for the Blocksworld effect models after pruning.

Action	Remaining	Total
`unstack`	8	230
`stack`	8	230
`putdown`	4	202
`pickup`	4	230
Total	**24**	**892**

The following are the extracted effect descriptions for the Blocksworld actions `unstack`, `stack`, `putdown`, and `pickup`. Here, `arg_0` denotes the action's first argument, and `object` is the concept containing every object. The term $\exists$`on.arg_0` denotes max-aggregated role quantification. The role $C \triangleright \top_R$ contains all pairs whose first object satisfies C; its inverse, $(C \triangleright \top_R)^-$, contains all pairs whose second object satisfies C, these correspond to the results from the Concept Extension relational operation of the model. For each candidate atom, the corresponding weighted expression is evaluated on the state and action arguments; the atom is added or deleted exactly when its score is strictly positive. A zero expression denotes an empty effect channel.

Unstack.

Add concepts: $\texttt{clear} \leftarrow 4.4879 \cdot \texttt{arg_0}$,
$\texttt{handempty} \leftarrow 0$,
$\texttt{holding} \leftarrow 3.7799 \cdot \exists \texttt{on}.\texttt{arg_0}$,
$\texttt{ontable} \leftarrow 0$,
Add roles: $\texttt{on} \leftarrow 0$;
Delete concepts: $\texttt{clear} \leftarrow 3.8084 \cdot \exists \texttt{on}.\texttt{arg_0}$,
$\texttt{handempty} \leftarrow 1.3133 \cdot \texttt{object}$,
$\texttt{holding} \leftarrow 0$,
$\texttt{ontable} \leftarrow 0$;
Delete roles: $\texttt{on} \leftarrow 2.3032 \cdot \texttt{on} - 1.3160 \cdot (\texttt{handempty} \triangleright \top_R)$
$- 1.2360 \cdot (\texttt{object} \triangleright \top_R)^- + 2.1723 \cdot (\texttt{arg_0} \triangleright \top_R)^-$.

Stack.

Add concepts: $\texttt{clear} \leftarrow 2.9644 \cdot \texttt{holding}$,
$\texttt{handempty} \leftarrow 1.7499 \cdot \texttt{object}$,
$\texttt{holding} \leftarrow 0$,
$\texttt{ontable} \leftarrow 0$;
Add roles: $\texttt{on} \leftarrow 1.8179 \cdot (\texttt{holding} \triangleright \top_R) - 1.5587 \cdot (\texttt{object} \triangleright \top_R)$
$- 1.4293 \cdot (\texttt{object} \triangleright \top_R)^- + 2.4469 \cdot (\texttt{arg_0} \triangleright \top_R)^-$;
Delete concepts: $\texttt{clear} \leftarrow 3.8027 \cdot \texttt{arg_0}$,
$\texttt{handempty} \leftarrow 0$,
$\texttt{holding} \leftarrow 2.7598 \cdot \texttt{holding}$,
$\texttt{ontable} \leftarrow 0$;
Delete roles: $\texttt{on} \leftarrow 0$.

Putdown.

Add concepts: $\texttt{clear} \leftarrow 1.7201 \cdot \texttt{holding}$,
$\texttt{handempty} \leftarrow 1.0549 \cdot \texttt{object}$,
$\texttt{holding} \leftarrow 0$,
$\texttt{ontable} \leftarrow 1.6490 \cdot \texttt{holding}$;
Add roles: $\texttt{on} \leftarrow 0$;
Delete concepts: $\texttt{clear} \leftarrow 0$,
$\texttt{handempty} \leftarrow 0$,
$\texttt{holding} \leftarrow 1.2131 \cdot \texttt{holding}$,
$\texttt{ontable} \leftarrow 0$;
Delete roles: $\texttt{on} \leftarrow 0$.

Pickup.

Add concepts: $\texttt{clear} \leftarrow 0$,
$\texttt{handempty} \leftarrow 0$,
$\texttt{holding} \leftarrow 1.8490 \cdot \texttt{arg_0}$,
$\texttt{ontable} \leftarrow 0$;
Add roles: $\texttt{on} \leftarrow 0$;
Delete concepts: $\texttt{clear} \leftarrow 1.5988 \cdot \texttt{arg_0}$,
$\texttt{handempty} \leftarrow 0.9914 \cdot \texttt{object}$,
$\texttt{holding} \leftarrow 0$,
$\texttt{ontable} \leftarrow 1.6608 \cdot \texttt{arg_0}$;
Delete roles: $\texttt{on} \leftarrow 0$.