DIFFERENTIABLE INDUCTIVE LOGIC PROGRAMMING WITH UNBOUND LOGIC

Anonymous authors
Paper under double-blind review

ABSTRACT

Differentiable inductive logic programming is an emerging area where logical rules are learned by differentiable models. The fundamental principle behind existing methods is relaxing forward chaining, a symbolic deduction procedure, into a continuous space. However, these methods rely on bound truth values and/or selective fuzzy logic operators, which suffer from gradient vanishing or imbalanced gradient flow. To this end, we propose using unbound logic, where truth values are further relaxed into unbound real numbers and fuzzy logic operators are non-selective, which theoretically alleviates the local gradient issues. Experiments show that our method achieves superior performance on inductive logic programming tasks. In addition, we explore combining neural perception networks with our fully differentiable logical models, resulting in an end-to-end differentiable model that achieves comparable performance with non-end-to-end visual ILP methods.

1 INTRODUCTION

Inductive Logic Programming (ILP) (Muggleton & Raedt, 1994; Cropper & Dumancic, 2022; Zhang et al., 2024) lies at the intersection of symbolic AI and machine learning. In contrast to conventional symbolic AI systems, which rely on manually written logical programs, ILP aims to learn such programs automatically from data. The learned programs, typically represented as first-order rules, are human-readable, editable, and lifted: a first-order rule applies uniformly to all constants rather than to individual ones. ILP can generalize from a small amount of data and supports continual and transfer learning (Evans & Grefenstette, 2018), which makes it an attractive research direction.

Classical ILP systems learn rules through discrete search over a symbolic hypothesis space (Law et al., 2020; Cropper & Morel, 2021; Feldstein et al., 2024), but they can be brittle in the presence of noise and difficult at scale (Glanois et al., 2022). Differentiable ILP addresses this issue by relaxing rule learning into a continuous optimization problem (Evans & Grefenstette, 2018). In this paradigm, forward chaining, a standard deduction procedure in logic programming, is approximated by a differentiable computation graph, allowing gradients to propagate through the deduction process, which has enabled differentiable ILP models to be integrated with neural components. The differentiability also gives rise to new applications, such as reinforcement learning (Payani & Fekri, 2020; Zimmer et al., 2023) and differentiable rule learning from sequence inputs (Gao et al., 2025). In this work, we develop a differentiable ILP model and extend it to visual ILP, where rules are learned from images.

The differentiability of existing ILP models typically depends on fuzzy logic operators, i.e., relaxations of classical logic operators (Table 1). For example, conjunction may be implemented by a t-norm, such as the Gödel t-norm, Łukasiewicz t-norm, or product t-norm (Evans & Grefenstette, 2018). However, these relaxations introduce optimization pathologies. Maximum- and minimum-based operators are selective: during backpropagation, they route gradient information only through the selected input (Evans & Grefenstette, 2018). Product-based operators multiply truth values bounded in [0, 1], which can cause gradients to vanish (Wang et al., 2024), especially when the operators are applied to many arguments or when forward chaining involves many steps. As a result, the choice of fuzzy logic operators directly affects the trainability of differentiable ILP models.

To address these local gradient issues, we adapt Łukasiewicz unbound logic (Lu) (Cintula et al., 2022; Haniková & Jankovec, 2026; CINTULA et al., 2026) to logic programming and use it as the basis for differentiable ILP. Unlike standard fuzzy logics, unbound logic relaxes truth values to the entire real line rather than restricting them to $[0, 1]$. Its operators are non-selective and avoid

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.

Table 1: Comparison of fuzzy logics. Conjunction corresponds to a t-norm, and disjunction corresponds to a t-conorm. Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ denote the input of a (multi-ary) t-norm or t-conorm, D denote the domain, and $c \in D$ denote a constant. Let $s = \sum_{i=1}^{n} x_i$. Let ∇ denote the gradient operator, $\mathbf{e}_i$ the ith unit vector, and $\mathbf{1}$ the all-ones vector. LuHorn refers to our adaptation of Łukasiewicz unbound logic to logic programming, defined in Sec. 4.1.

Fuzzy Logic	truth values	truth definition	(strong) conjunction $\bigwedge_{i=1}^{n} x_i$	(strong) disjunction $\bigvee_{i=1}^{n} x_i$	existential quantifier $\exists$
Gödel	$[0,1]$	$=1$	$\min_i x_i$	$\max_i x_i$	$\max_{c\in D} \cdot$
Product	$[0,1]$	$=1$	$\prod_{i=1}^{n} x_i$	$1 - \prod_{i=1}^{n}(1-x_i)$	$\max_{c\in D} \cdot$
Łukasiewicz	$[0,1]$	$=1$	$\max(0, s-(n-1))$	$\min(1, s)$	$\max_{c\in D} \cdot$
Lu	$\mathbb{R}$	≥ 0	s	-	-
LuHorn	$\mathbb{R}$	≥ 0	s	s	$\sum_{c\in D} \cdot$

Fuzzy Logic	∇ conjunction	∇ disjunction	∇ existential quantifier
Gödel	$\mathbf{e}_{\arg\min_i x_i}$	$\mathbf{e}_{\arg\max_i x_i}$	$\mathbf{e}_{\arg\max_{c\in D} \cdot}$
Product	$\left(\prod_{j\neq i} x_j\right)_{i=1}^{n}$	$\left(\prod_{j\neq i}(1-x_j)\right)_{i=1}^{n}$	$\mathbf{e}_{\arg\max_{c\in D} \cdot}$
Łukasiewicz	$\mathbf{0}$ for $s < n-1$; $\mathbf{1}$ for $s > n-1$	$\mathbf{1}$ for $s < 1$; $\mathbf{0}$ for $s > 1$	$\mathbf{e}_{\arg\max_{c\in D} \cdot}$
Lu	$\mathbf{1}$	-	-
LuHorn	$\mathbf{1}$	$\mathbf{1}$	$\mathbf{1}$

products of bounded truth values, providing a more favorable gradient flow for learning logical rules. Since Łukasiewicz unbound logic was originally developed for first-order logic rather than logic programming, we formulate a forward-chaining-like procedure and prove that it can recover the least Herbrand model in conventional logic programming, resulting in a new unbound logic LuHorn. Building on this procedure, we introduce DiffLu, a differentiable ILP model that learns logic programs through continuous optimization.

We further introduce vDiffLu, an end-to-end differentiable model for visual ILP. Existing visual ILP systems commonly follow a two-stage pipeline: a perception model is trained to extract symbolic representations from images, and an ILP model is then trained on those intermediate representations (Shindo et al., 2023; 2024). This separation requires additional supervision and prevents the perception and reasoning components from being optimized jointly. In contrast, vDiffLu combines a neural perception network with DiffLu in a single differentiable architecture, enabling the model to learn visual concepts and logical rules simultaneously without intermediate annotations. Moreover, because both neural networks and DiffLu operate over unbounded real-valued representations, the interface between perception and reasoning is natural from an optimization perspective.

We evaluate DiffLu on standard differentiable ILP benchmarks (Evans & Grefenstette, 2018) and vDiffLu on visual ILP tasks based on Kandinsky patterns (Müller & Holzinger, 2021) and Clevr-Hans (Stammer et al., 2021). The results show that DiffLu outperforms prior differentiable ILP models on the ILP benchmarks, while vDiffLu achieves competitive performance compared with separately trained visual ILP systems. Our contributions are threefold: (1) we formulate LuHorn, an unbound logic for logic programming; (2) based on LuHorn, we propose DiffLu, a differentiable ILP model; and (3) based on DiffLu, we introduce vDiffLu, an end-to-end differentiable model for visual ILP that jointly learns perception and logical reasoning.

2 Related work

2.1 Differentiable inductive logic programming

Differentiable inductive logic programming is an emerging approach to explainable AI. Early work studied differentiable learning of logical rules under restricted rule languages or predefined rule

structures. NeuralLP (Yang et al., 2017) and RNNLogic (Qu et al., 2021) consider only differentiable chain-like rules. Difflog (Si et al., 2019) assigns learnable weights to each clause in a predefined candidate rule set. LRI (Campero et al., 2018), end-to-end differentiable proving (Rocktäschel & Riedel, 2017), and soft-unification-based probabilistic logic models (Maene & Raedt, 2023) support differentiable logical reasoning through continuous predicate matching, but depend on predefined rule templates. Another line of work, including ∂ILP-ST (Shindo et al., 2021), αILP (Shindo et al., 2023), and NEUMANN (Shindo et al., 2024), uses a two-stage learning paradigm: first generating candidate clauses and then optimizing them. These approaches are not end-to-end differentiable and rely on a carefully specified initial hypothesis space.

Recent work has sought to reduce human intervention in differentiable ILP. ∂ILP (Evans & Grefenstette, 2018) uses parameterized rule templates to generate candidate rules, turning rule template design into parameter specification. However, ∂ILP requires detailed template definitions for each task, such as the number of existentially quantified variables for each intensional predicate. HRI (Glanois et al., 2022) introduces a theoretically complete set of proto-rules and the learning does not require prior structures. However, these two methods are restricted to rules with at most two body atoms. Moreover, ∂ILP suffers from the aforementioned gradient vanishing issue due to the product t-norm and HRI suffers from imbalanced gradient flow due to the Gödel t-norm. Another line of work, including D-LFIT (Gao et al., 2022) and DFORL (Gao et al., 2024a;b), learns from transitions rather than through full forward chaining. They use a product t-conorm as fuzzy disjunction, which suffers from gradient vanishing. Besides, DFORL (Gao et al., 2024b) allows target examples as input in the experiments, which may cause self-proved rules.[1] Our differentiable ILP model (DiffLu), closely related to ∂ILP and HRI, aims at alleviating these issues *caused by fuzzy logic operators* by introducing Łukasiewicz unbound logic (Sec. 2.3) into ILP. In addition, similar to HRI, our DiffLu does not require detailed task-specific rule specifications, i.e., only hyperparameters need to be adjusted for each task, which follows the recent proposal (Chen et al., 2025). Moreover, DiffLu can learn rules with many body atoms, allowing more expressive rule forms than those supported by ∂ILP and HRI.

2.2 Visual inductive logic programming

ILP has broad applications, such as knowledge base completion (Minervini et al., 2020; Qu et al., 2021; Cheng et al., 2023), bioinformatics (Srinivasan et al., 2025), sequence processing (Evans et al., 2021; 2022; Gao et al., 2025), games (Cropper et al., 2020), and reinforcement learning (Payani & Fekri, 2020; Zimmer et al., 2023). Our work focuses on visual ILP, which induces rules from images.

Visual ILP systems typically leverage neural perception networks to determine the symbolic groundings of images and use ILP to learn logical rules from the groundings. Early work is limited to simple images. Cingillioglu & Russo (2021) handles only pixelated image. Evans & Grefenstette (2018); Duan et al. (2023) handle only images in which each image (patch) corresponds to exactly one constant. On the other hand, αILP (Shindo et al., 2023) and NEUMANN (Shindo et al., 2024) can handle an image with several constants and identify the relations between the constants, fully exploiting the expressivity of first-order rules. NeSyGPT (Cunnington et al., 2024) replaces the neural perception network in αILP with a vision-language model, but at the cost of a substantially larger model and greater training requirements. Accordingly, our vDiffLu directly follows αILP and NEUMANN. Nonetheless, αILP, NEUMANN, and NeSyGPT all require the supervised training of the neural perception model with intermediate symbolic annotations. By contrast, our vDiffLu couples a neural perception network directly with DiffLu, forming an end-to-end differentiable joint model, which requires only end-to-end supervision. The use of unbounded truth values also provides a natural numerical interface between neural perception and logical reasoning.

2.3 Unbound logic

Conventional fuzzy logic typically uses truth values in $[0, 1]$ and the fuzzy logic operators ensure every logical formula is evaluated to a truth value in $[0, 1]$ (e.g., the first three rows in Table 1). Łukasiewicz unbound logic (Lu) is a recent variant of infinite-valued Łukasiewicz logic motivated by reasoning with vagueness and graded predicates (Cintula et al., 2022; Haniková & Jankovec, 2026; CINTULA et al., 2026). Lu keeps the same Łukasiewicz-style arithmetic but removes the bounding

[1] See, for example, the learned rules for Fizz and Buzz in their online supplementary materials.

operation and lets truth values range over the real line, which alleviates the gradient vanishing and imbalanced gradient problems. Lu was originally developped for conventional first-order logic; hence, we adapt it to logic programming, where the semantic model is restricted to the least Herbrand model.

3 BACKGROUND

3.1 LOGIC PROGRAMMING AND FORWARD CHAINING

Logic programming (Lloyd, 2012) is a declarative programming paradigm in which computation is expressed in terms of logical statements. A common formal setting for logic programming is Horn logic.[2] Horn *clauses* are written in the form $H \leftarrow B_1, \ldots, B_k,\ k \geq 0$. Each of $H, B_1, \ldots, B_k$ is in the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate and $t_1, \ldots, t_n$ are logical constants or logical variables. Variables that appear only in the body are called *existentially quantified variables*. $p(t_1, \ldots, t_n)$ is called a *ground atom* if all the $t_1, \ldots, t_n$ are constants. H is called the *head* atom of the rule, while the $B_1, \ldots, B_k$ are called *body* atoms. The clause is a *fact* if $k = 0$, while it is a *rule* if $k > 0$. A *logic program* $\mathcal{P}$ is a finite set of such facts and rules. Semantically, the program has a unique least Herbrand model I^*, the smallest set of ground atoms entailed by the program.

Forward chaining is a monotone inference procedure for deriving this least model. Starting from the facts in the program, forward chaining repeatedly applies rules whose bodies are satisfied by the facts derived so far. Formally, let I be a set of ground atoms. The immediate consequence operator (i.e., *transition*) $T_{\mathcal{P}}$ maps I to

$$T_{\mathcal{P}}(I) = \{\mathrm{head}(r) \mid c \in \mathcal{P}, r \in \mathrm{ground}(c), \mathrm{body}(r) \subset I\}, \tag{1}$$

where $\mathrm{ground}(c)$ denotes the set of ground clauses obtained by substituting all the variables in clause c with constants. Forward chaining computes the least fixed point of $T_{\mathcal{P}}$ by iterating from the empty interpretation:

$$I_0 = \varnothing, \qquad I_{n+1} = I_n \cup T_{\mathcal{P}}(I_n). \tag{2}$$

The fixed point $\bar{I}$ satisfies $\bar{I} = \bar{I} \cup T_{\mathcal{P}}(\bar{I})$ and coincide with I^*.

3.2 INDUCTIVE LOGIC PROGRAMMING

Inductive Logic Programming (ILP) learns a logical *hypothesis* $\mathcal{H}$ from given data, including *background knowledge* $\mathcal{B}$ and *target examples* $\mathcal{E}^+, \mathcal{E}^-$. Together with the background knowledge, the hypothesis should entail all positive examples $\mathcal{E}^+$ and not entail any negative examples $\mathcal{E}^-$, i.e.,

$$\begin{aligned} \mathcal{B} \wedge \mathcal{H} &\models e^+, \quad \forall e^+ \in \mathcal{E}^+ \\ \mathcal{B} \wedge \mathcal{H} &\not\models e^-, \quad \forall e^- \in \mathcal{E}^- \end{aligned} \tag{3}$$

Horn clauses are typically used to represent $\mathcal{H}, \mathcal{B}, \mathcal{E}^+, \mathcal{E}^-$ (Nienhuys-Cheng & de Wolf, 1997). For example, *facts* $\mathcal{B} = \{Even(0), Even(2), Succ(0,1), Succ(1,2), Succ(2,3)\}$ define even numbers and the successor relation between natural numbers in the *domain* $\{0,1,2,3\}$ as background knowledge. To learn the concept of odd numbers, the positive target examples should be $\mathcal{E}^+ = \{Odd(1), Odd(3)\}$ and the negative target examples should be $\mathcal{E}^- = \{Odd(0), Odd(2)\}$. An ILP method may learn the *rule* $\mathcal{H} = Odd(Y) \leftarrow Even(X) \wedge Succ(X,Y)$. *Unground atoms* are exemplified by $Odd(Y), Even(X), Succ(X,Y)$, which are formed from predicates and logical variables. An *intensional* predicate is defined by rules instead of being defined by the facts in $\mathcal{B}$. In ILP, intensional predicates refer to the target predicate.[3]

To confirm a hypothesis $\mathcal{H}$, forward chaining can be used by finding the least Herbrand model. $\mathcal{E}^+$ should be a subset of the least Herbrand model and $\mathcal{E}^-$ should be disjoint from it.

[2] Following Evans & Grefenstette (2018); Glanois et al. (2022); Gao et al. (2024b), we focus on the function-free fragment of Horn logic with predicate arities of at most 2. Functions can be replaced with predicates. Besides, DiffLu can be straightforwardly extended to higher arities.

[3] Auxiliary predicates are also intensional (Evans & Grefenstette, 2018). Our DiffLu can be directly extended to support auxiliary predicates, but we omit them for simplicity of presentation.

3.3 FORWARD CHAINING AND FUZZY LOGIC OPERATORS

To see how fuzzy logic operators participate in the forward chaining procedure for differentiable ILP, we first abstract it into a numerical computation procedure. Define the initial valuation function $v^{(0)} : \mathrm{HB} \mapsto \mathrm{TV}$, where HB is the Herbrand base, the set of all ground atoms, and TV is the range of truth values. In conventional forward chaining, $\mathrm{TV} \triangleq \{0, 1\}$ and $v^{(0)}(a) = 1$ if and only if $a \in \mathcal{B}$. One step of forward chaining defined in Sec. 3.1 can be reformulated as

$$v^{(n+1)}(a) \quad \longleftarrow \quad \mathrm{S}\left(v^{(n)}(a), \quad \mathop{\mathrm{R}}_{c\in\mathcal{P}} \mathop{\exists}_{\substack{r\in\mathrm{ground}(c),\\ \mathrm{head}(r)=a}} \bigwedge_{b\in\mathrm{body}(r)} v^{(n)}(b)\right) \tag{4}$$

$\mathrm{S}, \mathrm{R}, \exists, \wedge$ are aggregation functions. The latter two correspond to the existential quantifier (corresponding to $r \in \mathrm{ground}(c)$ in Eq. 1) and conjunction. R aggregates the valuations from different clauses in the program, referring to $c \in \mathcal{P}$ of Eq. 1. S aggregates the valuations from two consecutive steps, corresponding to the union in Eq. 2. Besides, S, R are usually interpreted as disjunction or maximum. The forward chaining procedure defined by Eq. 4 terminates when the valuation function remains unchanged in a certain step (i.e., fixed point attained).

In classical forward chaining, the conjunction can be implemented with any of the Gödel, product, Łukasiewicz operators, since these three fuzzy logic systems coincide with classical logic when $\mathrm{TV} = \{0, 1\}$. $\mathrm{S}, \mathrm{R}, \exists$ are naturally set to $\max$. The fixed point of v then indicates whether an atom can be derived through forward chaining. It is also worth mentioning that, if we set $\mathrm{S}, \mathrm{R}, \exists$ all to $\sum$, the fixed point of v indicates the number of possibly duplicated derivation paths for an atom, which our LuHorn uses (Sec. 4.1). However, the truth values may be non Boolean for two reasons. One is that differentiable ILP models use continuous weights to relax rules (e.g., Evans & Grefenstette (2018); Glanois et al. (2022)). Another reason is that, in visual ILP, the truth values of the background facts are computed via a neural perception network, whose outputs are typically real numbers (Shindo et al., 2023; 2024). Therefore, fuzzy logic operators are involved.

In addition, if we define the aggregation functions by differentiable operators, the numerical forward chaining defines an end-to-end differentiable computation graph, which is the essential idea of how existing differentiable ILP models achieve differentiability (Evans & Grefenstette, 2018; Campero et al., 2018; Glanois et al., 2022). However, Table 1 shows that Gödel and Łukasiewicz operators cause an imbalanced gradient problem (Evans & Grefenstette, 2018), and product conjunction and disjunction cause a gradient vanishing problem (Wang et al., 2024).[4]

4 METHODS

In this section, we introduce our proposed unbound logic programming system LuHorn (Sec. 4.1), unbound differentiable ILP model DiffLu (Sec. 4.2), and joint visual ILP model vDiffLu (Sec. 4.3).

4.1 LUHORN: ŁUKASIEWICZ UNBOUND LOGIC FOR HORN RULES

The recently proposed first-order fuzzy logic system Łukasiewicz unbound logic (Lu) (Cintula et al., 2022; Haniková & Jankovec, 2026; CINTULA et al., 2026) avoids the aforementioned issues. Lu drops the maximum operators from Łukasiewicz conjunction and extends the range of truth values to the entire real line, enabling gradient information to be sent to all its inputs. In this section, we develop LuHorn, which adapts Lu to logic programming; in particular, we propose a forward-chaining-like procedure with LuHorn operators, which can also find the conventional least Herbrand model.

More specifically, we consider the shifted version of Lu proposed in (CINTULA et al., 2026), which shifts the truth values of Lu by 1.[5] Shifted Lu lets 0 be the boundary between *true* and *false*, and a larger truth value indicates a higher degree of truth. We argue that the unbounded truth space and the symmetry of truth align with the characteristics of the neural model weights, which may help differentiable learning via gradient-based optimizers. The conjunction of shifted Lu is simply the addition of its conjuncts (CINTULA et al., 2026), which follows the residuation law between (strong)

[4] van Krieken et al. (2022) considers more fuzzy logic operators, which also have the same issues.

[5] Table 1 exhibits the shifted version.

conjunction and implication, one of the critical laws for constructing fuzzy logic systems (Cintula et al., 2011; 2022).

Regarding $\mathrm{S}, \mathrm{R}, \exists$, to avoid bound or selective operators, we simply set them all to sum. Recall that, in classical forward chaining, this corresponds to path counting. We will justify these choices in DiffLu. Therefore, the forward chaining procedure corresponding to LuHorn is characterized by

$$v^{(n+1)}(a) \quad \longleftarrow \quad v^{(n)}(a) \quad + \quad \sum_{c \in \mathcal{P}} \sum_{\substack{r \in \mathrm{ground}(c), \\ \mathrm{head}(r)=a}} \sigma\left(\mathrm{body}(r)\right) \cdot \sum_{b \in \mathrm{body}(r)} v^{(n)}(b), \tag{5}$$

where $\mathrm{TV} \triangleq \mathbb{R}$. We set $v^{(0)}(a)$ to 1 if $a \in \mathcal{B}$ and -1 otherwise. Since Lu does not preserve falsity (Appendix A.6), we add a hard gate function $\sigma\left(\mathrm{body}(r)\right) = \mathbf{1}[\min_{b \in \mathrm{body}(r)} v(b) >= 0]$ indicating whether the ground rule r is fired. From the gradient analysis in Table 1, the differentiable computation graph defined by Eq. 5 does not contain operators that are prone to vanish gradients or block backward gradients, except for the gate function, which will be alleviated in DiffLu. The following theorem states that LuHorn's forward-chaining procedure can exactly recover the classical least Herbrand model, which is proved in A.

Theorem 4.1 (Soundness and Completeness)**.** *The forward-chaining-like procedure defined by Eq. 5 is sound and complete with respect to the classical least Herbrand model.*

Intuitively, the gates ensure that a rule is fired *iff.* all the bodys are *true*, which coincides with classical forward chaining. Refer to Algorithm 1 in Appendix A for a pseudo-code.

4.2 DIFFLU: AN END-TO-END DIFFERENTIABLE ILP SYSTEM

Recall that differentiable ILP typically relaxes a forward-chaining procedure and assigns weights to learn a program hypothesis. The unbound logic programming system LuHorn is already equipped with a forward-chaining procedure. Now it remains to continuously parameterize Horn rules and integrate them into the forward-chaining. Our relaxation is highly straightforward, specified below.

Let $\mathcal{F}$ be the set of first-order features (Gao et al., 2024b) of the set of all the predicates involved in an ILP task with at most V logical variables. For example, $\{q(X,Y), q(Y,X), q(X,Z), q(Z,X), q(Y,Z), q(Z,Y)\}$ is the feature set for the predicate set $\{q\}$ with at most three variables. Let $\mathcal{Q}$ be all the intensional predicates involved in an ILP task. For each intensional predicate $p \in \mathcal{Q}$ defined by rule $c^{(i)}$, we assign a learnable weight $|w_p^{(i)}(f)|$ for each feature $f \in \mathcal{F}$, indicating the degree to which this feature belongs to c's body. The superscript i indicates that each p can be defined by more than one rule. Since the valuation of a rule is defined as the difference between the head and body (CINTULA et al., 2026), we define for each ground rule $r \in \mathrm{ground}(c^{(i)})$ the valuation of its head with Lu's conjunction

$$v(\mathrm{head}(r)) \longleftarrow \sum_{f \in \mathcal{F}} |w_p^{(i)}(f)| \cdot v(f_r), \tag{6}$$

where f_r is the ground feature under the same substitution for r. Eq. 6 ensures the learnable rule is always evaluated to *true*. When $w_p^{(i)}(f)$ is zero, the valuation of the ground feature $v(f_r)$ does not affect the head's valuation, i.e., the feature f does not belong to rule $c^{(i)}$'s body. The unboundness of the learnable weights aligns with the unboundness of LuHorn. By integrating Eq. 6 into Eq. 5, we establish a differentiable and learnable forward chaining procedure as follows

$$v^{(n+1)}(a) \quad \longleftarrow \quad v^{(n)}(a) \quad + \quad \sum_{p \in \mathcal{Q}, i \in [\![R]\!]} \sum_{\substack{r \in \mathrm{ground}(c_p^{(i)}), \\ \mathrm{head}(r)=a}} \sum_{f \in \mathcal{F}} |w_p^{(i)}(f)| \cdot v(f_r), \tag{7}$$

where R denotes the number of rules learned for each intensional predicate.

The gates in Eq. 5 are absorbed into the learnable weights. Intuitively, a well-learned weight set is expected to make $v^{(n+1)}(a)$ evaluated to positive if all the body atoms are evaluated to *true*, and negative otherwise.

The minimization objective is

$$[1 - \min(v^*(a), 1)] \cdot \mathbf{1}[\hat{v}(a) \geq 0] + [\max(v^*(a), -1) - (-1)] \cdot \mathbf{1}[\hat{v}(a) < 0] \tag{8}$$

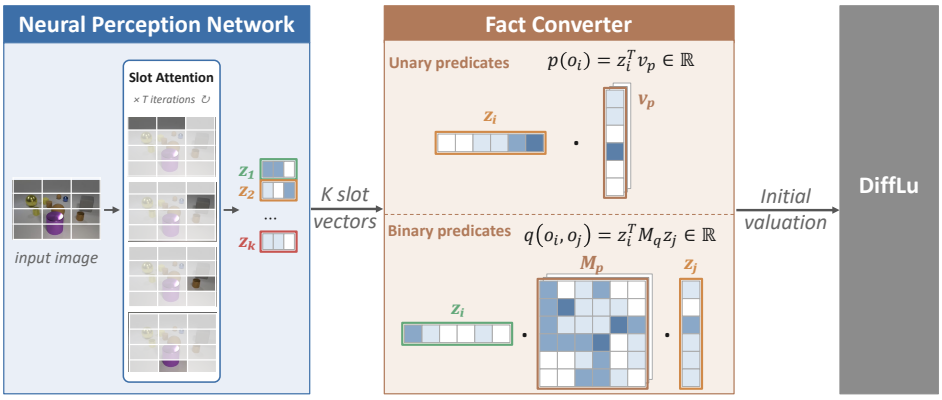


Figure 1: Overview of **vDiffLu**, an end-to-end differentiable joint architecture for visual ILP. $z_1, \ldots, z_k$ are embeddings for constants $o_1, \ldots, o_k$ identified by neural perception network. p represents a unary predicate and q represents a binary predicate. v_p, M_p are trainable matrices.

This objective drives the fixed-point valuation $v^*(a)$ to 1 if it should be *true*, and to -1 otherwise. The cutoff $(\min, \max)$ in the objectives is to avoid over-training if the corresponding valuation is already correct enough. Besides, the path-counting-like operator choice in Sec. 4.1 fits this objective well. Intuitively, a positive valuation should correspond to at least one path of derivation, while a negative valuation should correspond to zero path of derivation.

After training, we use a heuristic approach to extract symbolic rules from parameterized rules (Eq. 6), where the features with relatively large absolute weights are retained. (Appendix B)

4.3 vDiffLu: An end-to-end differentiable joint learning model

With DiffLu, an end-to-end differentiable ILP model working in full real number space, designing an end-to-end differentiable visual ILP model becomes easy. We propose vDiffLu (Fig. 1), a differentiable joint model that simultaneously learns a neural perception network and an ILP model. Similar to αILP (Shindo et al., 2023) and NEUMANN (Shindo et al., 2024), we use slot attention (Locatello et al., 2020) to identify K slots from the input image and use a fact converter to compute the valuations of background knowledge. The slots refer to objects since αILP and NEUMANN pretrain slot attention with object annotations. In contrast, vDiffLu has no pretraining; hence, the slots are interpreted just as logical constants.

Besides, our fact converter is more universal: the valuation of unary predicates is obtained by computing the inner product of a slot's vector and the predicates' vectors, while the valuation of binary predicates is obtained by computing the bilinear score between two slots with the predicates' matrix. Different predicates are assigned a separate learnable vector/matrix. We mainly need to set the number of unary predicates and binary predicates to be learned. In contrast, the fact converter used in αILP and NEUMANN is more bound to specific predicates and is hard to extend to other predicates. After computing the initial valuation from the fact converter, DiffLu takes it as input and computes the final valuations iteratively. The training objective is the same as Eq. 8.

5 Experiments

In experiments, we first evaluate DiffLu on standard ILP benchmarks to confirm that our proposed unbound fuzzy logic system benefits differentiable ILP. Second, we evaluate on visual ILP benchmarks to justify that vDiffLu achieves comparable performance with separately trained visual ILP systems.

5.1 Differentiable ILP

We conduct experiments on the standard benchmarks designed for differentiable ILP Evans & Grefenstette (2018). The dataset contains 17 ILP tasks, containing graph relations, arithmetic, list

Table 2: Results for differentiable ILP. †These methods are not directly comparable (see Sec. 2.1).

Model	Predec	LessTh	Adjac	Related	Memb	Conn	EvenSu	Grandp	Son
NeuralLP	100	100	0	0	0	0	0	100	0
∂ILP †	100	100	50.5	100	100	100	48.5	96.5	100
LRI †	100	100	100	100	100	100	100	100	100
HRI	100	100	100	100	100	100	40	100	100
DFORL †	100	100	100	100	100	100	0	100	100
DiffLu	100	100	100	100	100	100	100	100	100

Model	Undirect	EvenOdd	Cyclic	TwoC	Color	Len	Buzz	Fizz	Avg.
NeuralLP	0	0	0	0	0	0	0	0	17.6
∂ILP †	100	100	100	95	94.5	92.5	35	10	83.7
LRI †	100	100	100	0	0	100	70	10	81.2
HRI	100	40	100	100	100	0	40	0	77.6
DFORL †	100	0	100	100	100	0	0	0	70.6
DiffLu	100	100	0	100	100	0	90	40	**84.1**

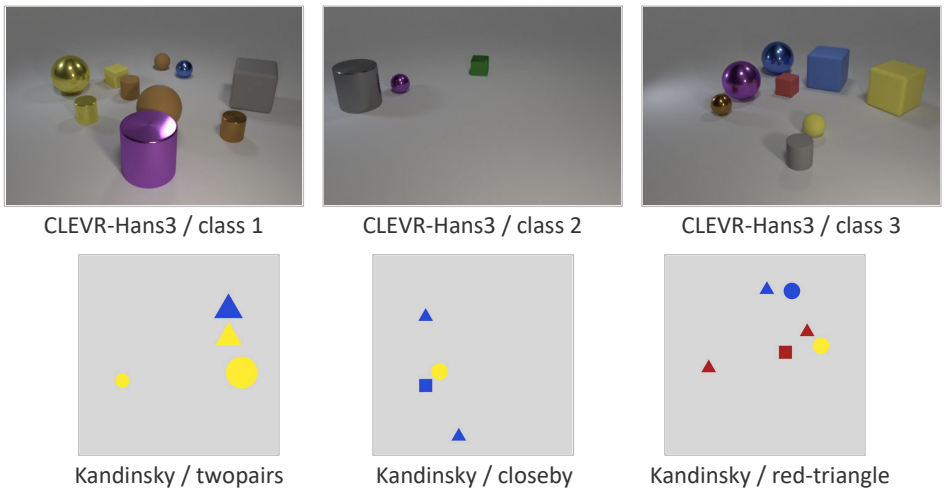


Figure 2: Visual ILP datasets. The task is to classify visual scenes by obtaining explicit classification rules. Each CLEVR-Hans3 pattern is characterized as follows: (class 0) A large cube and a large cylinder. (class 1) A small metal cube and a small sphere. (class 2) A large blue sphere and a small yellow sphere. Each Kandinsky pattern is characterized as follows: (twopairs) A pair with the same shape and color, and another pair with the same shape and different colors. (closeby) Two objects closely located. (red-triangle) A red triangle close to a non-triangle and non-red object.

relations, and family trees. We already present an example task in Sec. 3.2. Table 2 shows the results on these datasets. Following previous works, the reported metric is the success rate over at least 10 random seeds. A run is considered successful if, on the validation set, the mean squared error between the truth value of the target examples and the predicted valuations is less than 1e-4. Before evaluation, we do the following transformation to the output of DiffLu $(\min(\max(-1, v), 1) + 1)/2$, which aligns DiffLu's truth value space within $[0, 1]$ (the conventional space used by baselines).

Notably, DiffLu outperforms HRI, the best comparable baseline, by 6.5 percentage points. In addition, DiffLu brings notable improvement on Fizz, the most difficult task involving complicated recursive rules (e.g., $Fizz(W) \leftarrow Fizz(X) \wedge succ(X, Y) \wedge succ(Y, Z) \wedge succ(Z, W)$), confirming the benefits of DiffLu's ability to learn many-body-atom rules (i.e., relatively longer rules). Refer to Appendix D for failure analysis.

Table 3: Results for Visual ILP.

Model	closeby-2k	red triangle	two pairs	Avg.
CNN	52.22 ± 0.35	51.73 ± 0.65	49.91 ± 0.29	51.30
YOLO	59.96 ± 0.49	64.94 ± 0.40	90.83 ± 1.48	71.91
αILP	81.21 ± 0.00	$\mathbf{99.98 \pm 0.00}$	$\mathbf{99.98 \pm 0.00}$	**93.72**
NEUMANN	79.76 ± 24.69	54.68 ± 3.07	51.21 ± 2.49	61.89
vDiffLu	$\mathbf{99.99 \pm 0.01}$	$\mathbf{99.87 \pm 0.02}$	63.06 ± 0.62	87.64
Model	**CH1**	**CH2**	**CH3**	**Avg.**
CNN	70.32 ± 0.74	68.06 ± 1.52	70.07 ± 0.74	69.48
NeSy	83.02 ± 0.09	80.33 ± 0.26	89.92 ± 0.19	84.42
αILP	$\mathbf{97.27 \pm 0.26}$	$\mathbf{96.61 \pm 0.20}$	$\mathbf{98.27 \pm 0.20}$	**97.38**
NEUMANN	80.3 ± 7.9	66.7 ± 0.02	82.4 ± 9.7	76.47
vDiffLu	$\mathbf{96.43 \pm 0.50}$	$\mathbf{93.69 \pm 1.46}$	$\mathbf{95.53 \pm 3.81}$	**95.22**

5.2 VISUAL ILP

Following αILP and NEUMANN, we evaluate vDiffLu on Kandinsky Patterns (Müller & Holzinger, 2021), which probes relational generalisation over abstract scenes, and CLEVR-Hans3 (Stammer et al., 2021), which learns rule-based classifier on photorealistic 3D scenes. **Kandinsky Patterns.** We use the three binary-classification patterns, namely `twopairs`, `closeby`, `red-triangle`. Each problem contains $1\text{k}/1\text{k}$ positive/negative training images, $5\text{k}/5\text{k}$ validation, and $5\text{k}/5\text{k}$ test (22k images per pattern). Source images are 640×640 RGBA and are produced by the official `kandinsky_generator`.[6] The transform pipeline is the same as αILP and NEUMANN and is documented in Appendix C.1. **CLEVR-Hans3.** In each of the three binary classification problems, we have $3000/6000$ train and $750/1500$ validation images. We do not consider the test split since it is for studying confounding factors, which is out of the scope of this work.

Following αILP and NEUMANN, we compare against five baselines, covering pure neural classifiers (CNN, YOLO), neural perception combined with a learned predictor (NeSy), and visual ILP systems (αILP, NEUMANN). We rerun the open-source code for all the baselines provided in the repository of αILP or NEUMANN, with the same hyperparameters specified in the original paper or repository. In Table 3, we report the average and standard deviation over 5 random seeds. The results show that vDiffLu outperforms neural baselines. In addition, vDiffLu achieves performance comparable to αILP and NEUMANN, where the neural perception network is trained separately with intermediate annotations. Although αILP can outperform vDiffLu, it is suffered from severe memory consumption (Shindo et al., 2024). In contrast, vDiffLu is trained end-to-end without extra annotations with small memory consumption ($\sim 5\%$ αILP).

6 CONCLUSION

We introduced LuHorn, an adaptation of Łukasiewicz unbound logic to Horn logic programming, and DiffLu, a differentiable ILP model built on unbounded truth values and non-selective additive operators. This approach alleviates local gradient vanishing and imbalanced gradient flow, while LuHorn's gated forward-chaining procedure recovers the classical least Herbrand model. On standard ILP benchmarks, DiffLu improves the average success rate over the strongest directly comparable baseline by 6.5 percentage points and demonstrates the benefits of learning rules with many body atoms. We further introduced vDiffLu, which jointly learns neural perception and logical reasoning from images, achieving competitive visual ILP performance without intermediate annotations. Together, these results highlight unbound logic as a promising foundation for differentiable rule learning and end-to-end neuro-symbolic modeling. We comprehensively discuss the **limitations and future works** in Appendix D.

[6] `https://github.com/ml-research/kandinsky_generator`

AI USE STATEMENT

In this work, we used generative AI tools for assisting in the writing of proofs and assisting with translation. We have not used generative AI tools for helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, implementing methods, supporting qualitative and thematic data analysis, and interpreting results. Generating synthetic data sets and cleaning and reformating dataset are not applicable to this work. Additionally, we used generative AI tools for summarizing or analysing existing literature, sourcing/searching for information, and identifying relevant literature. We have reviewed all AI-assisted work. We checked the correctness of proofs improved by LLMs. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We will publicly release the code when publishing this manuscript.

REFERENCES

Andres Campero, Aldo Pareja, Tim Klinger, Josh Tenenbaum, and Sebastian Riedel. Logical rule induction and theory learning using neural theorem proving. *CoRR*, abs/1809.02193, 2018. URL http://arxiv.org/abs/1809.02193.

Si Chen, Richong Zhang, and Xu Zhang. Geoilp: A synthetic dataset to guide large-scale rule induction. In *The Thirteenth International Conference on Learning Representations*, 2025.

Kewei Cheng, Nesreen K. Ahmed, and Yizhou Sun. Neural compositional rule learning for knowledge graph reasoning. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=F8VKQyDgRVj.

Nuri Cingillioglu and Alessandra Russo. pix2rule: End-to-end neuro-symbolic rule learning. In Artur S. d'Avila Garcez and Ernesto Jiménez-Ruiz (eds.), *Proceedings of the 15th International Workshop on Neural-Symbolic Learning and Reasoning as part of the 1st International Joint Conference on Learning & Reasoning (IJCLR 2021), Virtual conference, October 25-27, 2021*, CEUR Workshop Proceedings, pp. 15–56. CEUR-WS.org, 2021. URL https://ceur-ws.org/Vol-2986/paper3.pdf.

Petr Cintula, Petr Hájek, Carles Noguera, et al. *Handbook of Mathematical Fuzzy Logic. Volume 1*, volume 37. College publications, 2011.

Petr Cintula, Berta Grimau, Carles Noguera, and Nicholas J. J. Smith. These degrees go to eleven: fuzzy logics and gradable predicates. *Synthese*, 200(6):445, Oct 2022. ISSN 1573-0964. doi: 10.1007/s11229-022-03909-2. URL https://doi.org/10.1007/s11229-022-03909-2.

PETR CINTULA, FILIP JANKOVEC, and CARLES NOGUERA. Superabelian logics. *The Review of Symbolic Logic*, pp. 1–27, March 2026. ISSN 1755-0211. doi: 10.1017/s1755020326101129. URL http://dx.doi.org/10.1017/S1755020326101129.

Andrew Cropper and Sebastijan Dumancic. Inductive logic programming at 30: A new introduction. *J. Artif. Intell. Res.*, 74:765–850, 2022. doi: 10.1613/JAIR.1.13507. URL https://doi.org/10.1613/jair.1.13507.

Andrew Cropper and Rolf Morel. Predicate invention by learning from failures. *arXiv preprint arXiv:2104.14426*, 2021.

Andrew Cropper, Richard Evans, and Mark Law. Inductive general game playing. *Mach. Learn.*, 109(7):1393–1434, 2020. doi: 10.1007/S10994-019-05843-W. URL https://doi.org/10.1007/s10994-019-05843-w.

Daniel Cunnington, Mark Law, Jorge Lobo, and Alessandra Russo. The role of foundation models in neuro-symbolic learning and reasoning. In Tarek R. Besold, Artur d'Avila Garcez, Ernesto Jiménez-Ruiz, Roberto Confalonieri, Pranava Madhyastha, and Benedikt Wagner (eds.), *Neural-Symbolic Learning and Reasoning - 18th International Conference, NeSy 2024, Barcelona, Spain, September 9-12, 2024, Proceedings, Part I*, Lecture Notes in Computer Science, pp. 84–100. Springer, 2024. doi: 10.1007/978-3-031-71167-1_5. URL https://doi.org/10.1007/978-3-031-71167-1_5.

Xuguang Duan, Xin Wang, Peilin Zhao, Guangyao Shen, and Wenwu Zhu. Deeplogic: Joint learning of neural perception and logical reasoning. *IEEE Trans. Pattern Anal. Mach. Intell.*, 45(4): 4321–4334, 2023. doi: 10.1109/TPAMI.2022.3191093. URL https://doi.org/10.1109/TPAMI.2022.3191093.

Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *J. Artif. Intell. Res.*, 61:1–64, 2018. doi: 10.1613/JAIR.5714. URL https://doi.org/10.1613/jair.5714.

Richard Evans, José Hernández-Orallo, Johannes Welbl, Pushmeet Kohli, and Marek Sergot. Making sense of sensory input. *Artificial Intelligence*, 293:103438, 2021.

Richard Evans, Matko Bosnjak, Lars Buesing, Kevin Ellis, David Pfau, Pushmeet Kohli, and Marek J Sergot. Making sense of raw input. In *IJCAI*, 2022.

Jonathan Feldstein, Dominic Phillips, and Efthymia Tsamoura. Efficiently learning probabilistic logical models by cheaply ranking mined rules. *CoRR*, abs/2409.16238, 2024. doi: 10.48550/ARXIV.2409.16238. URL https://doi.org/10.48550/arXiv.2409.16238.

Kun Gao, Hanpin Wang, Yongzhi Cao, and Katsumi Inoue. Learning from interpretation transition using differentiable logic programming semantics. *Mach. Learn.*, 111(1):123–145, 2022. doi: 10.1007/S10994-021-06058-8. URL https://doi.org/10.1007/s10994-021-06058-8.

Kun Gao, Katsumi Inoue, Yongzhi Cao, and Hanpin Wang. A differentiable first-order rule learner for inductive logic programming (abstract reprint). In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI 2024, Jeju, South Korea, August 3-9, 2024*, pp. 8478. ijcai.org, 2024a. URL https://www.ijcai.org/proceedings/2024/950.

Kun Gao, Katsumi Inoue, Yongzhi Cao, and Hanpin Wang. A differentiable first-order rule learner for inductive logic programming. *Artif. Intell.*, 331:104108, 2024b. doi: 10.1016/J.ARTINT.2024.104108. URL https://doi.org/10.1016/j.artint.2024.104108.

Kun Gao, Katsumi Inoue, Yongzhi Cao, Hanpin Wang, and Yang Feng. Differentiable rule induction from raw sequence inputs. In *The Thirteenth International Conference on Learning Representations*, 2025.

Claire Glanois, Zhaohui Jiang, Xuening Feng, Paul Weng, Matthieu Zimmer, Dong Li, Wulong Liu, and Jianye Hao. Neuro-symbolic hierarchical rule induction. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (eds.), *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pp. 7583–7615. PMLR, 2022. URL https://proceedings.mlr.press/v162/glanois22a.html.

Zuzana Haniková and Filip Jankovec. Satisfiability in Łukasiewicz logic and its unbounded relative. volume 363, pp. 14:1–14:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2026. doi: 10.4230/LIPICS.CSL.2026.14. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.CSL.2026.14.

Mark Law, Alessandra Russo, and Krysia Broda. The ILASP system for inductive learning of answer set programs. *CoRR*, abs/2005.00904, 2020. URL https://arxiv.org/abs/2005.00904.

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

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. *Advances in neural information processing systems*, 33:11525–11538, 2020.

Jaron Maene and Luc De Raedt. Soft-unification in deep probabilistic logic. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

Pasquale Minervini, Sebastian Riedel, Pontus Stenetorp, Edward Grefenstette, and Tim Rocktäschel. Learning reasoning strategies in end-to-end differentiable proving. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, Proceedings of Machine Learning Research, pp. 6938–6949. PMLR, 2020. URL http://proceedings.mlr.press/v119/minervini20a.html.

Stephen H. Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *J. Log. Program.*, 19/20:629–679, 1994. doi: 10.1016/0743-1066(94)90035-3. URL https://doi.org/10.1016/0743-1066(94)90035-3.

Heimo Müller and Andreas Holzinger. Kandinsky patterns. *Artif. Intell.*, 300:103546, 2021. doi: 10.1016/J.ARTINT.2021.103546. URL https://doi.org/10.1016/j.artint.2021.103546.

Shan-Hwei Nienhuys-Cheng and Ronald de Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Computer Science*. Springer, 1997. ISBN 3-540-62927-0. doi: 10.1007/3-540-62927-0. URL https://doi.org/10.1007/3-540-62927-0.

Ali Payani and Faramarz Fekri. Incorporating relational background knowledge into reinforcement learning via differentiable inductive logic programming. *CoRR*, abs/2003.10386, 2020. URL https://arxiv.org/abs/2003.10386.

Meng Qu, Jun-Kun Chen, Louis-Pascal A. C. Xhonneux, Yoshua Bengio, and Jian Tang. Rnnlogic: Learning logic rules for reasoning on knowledge graphs. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=tGZu6DlbreV.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 3788–3800, 2017.

Hikaru Shindo, Masaaki Nishino, and Akihiro Yamamoto. Differentiable inductive logic programming for structured examples. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 5034–5041. AAAI Press, 2021. doi: 10.1609/AAAI.V35I6.16637. URL https://doi.org/10.1609/aaai.v35i6.16637.

Hikaru Shindo, Viktor Pfanschilling, Devendra Singh Dhami, and Kristian Kersting. αilp: thinking visual scenes as differentiable logic programs. *Mach. Learn.*, 112(5):1465–1497, 2023. doi: 10.1007/S10994-023-06320-1. URL https://doi.org/10.1007/s10994-023-06320-1.

Hikaru Shindo, Viktor Pfanschilling, Devendra Singh Dhami, and Kristian Kersting. Learning differentiable logic programs for abstract visual reasoning. *Mach. Learn.*, 113(11):8533–8584, 2024. doi: 10.1007/S10994-024-06610-2. URL https://doi.org/10.1007/s10994-024-06610-2.

Xujie Si, Mukund Raghothaman, Kihong Heo, and Mayur Naik. Synthesizing datalog programs using numerical relaxation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 6117–6124. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/847. URL https://doi.org/10.24963/ijcai.2019/847.

Ashwin Srinivasan, Tirtharaj Dash, A Baskar, Sanjay Kumar Dey, and Mainak Banerjee. Identifying a logical specification and a program for an llm-based generator of lead molecules. *bioRxiv*, 2025. doi: 10.1101/2025.02.14.634875. URL https://www.biorxiv.org/content/early/2025/02/16/2025.02.14.634875.

Wolfgang Stammer, Patrick Schramowski, and Kristian Kersting. Right for the right concept: Revising neuro-symbolic concepts by interacting with their explanations. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pp. 3619–3629. Computer Vision Foundation / IEEE, 2021. doi: 10.1109/CVPR46437.2021.00362. URL https://openaccess.thecvf.com/content/CVPR2021/html/Stammer_Right_for_the_Right_Concept_Revising_Neuro-Symbolic_Concepts_by_Interacting_CVPR_2021_paper.html.

Emile van Krieken, Erman Acar, and Frank van Harmelen. Analyzing differentiable fuzzy logic operators. *Artif. Intell.*, 302:103602, 2022. doi: 10.1016/J.ARTINT.2021.103602. URL https://doi.org/10.1016/j.artint.2021.103602.

Zhuo Wang, Wei Zhang, Ning Liu, and Jianyong Wang. Learning interpretable rules for scalable data representation and classification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 46(2):1121–1133, 2024. doi: 10.1109/TPAMI.2023.3328881. URL https://doi.org/10.1109/TPAMI.2023.3328881.

Fan Yang, Zhilin Yang, and William W Cohen. Differentiable learning of logical rules for knowledge base reasoning. *Advances in neural information processing systems*, 30, 2017.

Zheng Zhang, Levent Onder Yilmaz, and Bo Liu. A critical review of inductive logic programming techniques for explainable AI. *IEEE Trans. Neural Networks Learn. Syst.*, 35(8):10220–10236, 2024. doi: 10.1109/TNNLS.2023.3246980. URL https://doi.org/10.1109/TNNLS.2023.3246980.

Matthieu Zimmer, Xuening Feng, Claire Glanois, Zhaohui Jiang, Jianyi Zhang, Paul Weng, Dong Li, Jianye Hao, and Wulong Liu. Differentiable logic machines. *Trans. Mach. Learn. Res.*, 2023, 2023. URL https://openreview.net/forum?id=mXfkKtu5JA.

A FORWARD CHAINING FOR ŁUKASIEWICZ UNBOUND LOGIC

In this appendix, we prove the soundness and completeness of the LuHorn procedure in Eq. 5 with respect to the classical least Herbrand model. We restate the necessary concepts and write v_n for $v^{(n)}$ in the main text.

In the following, we describe a forward-chaining procedure for definite Horn programs whose quantitative layer is evaluated in the shifted version of Łukasiewicz unbound logic. The procedure deliberately separates two roles that coincide in ordinary Boolean forward chaining: rule applicability and head-value propagation. Rule applicability is controlled by a gate based on the lattice meet, whereas each fired rule adds the sum of its body values to its head's current valuation.

A.1 HORN PROGRAMS AND HERBRAND MODELS

Let $\mathcal{L}$ be a first-order language and let HU_P and HB_P denote, respectively, the Herbrand universe and Herbrand base of a definite Horn program P. A ground definite Horn clause has the form

$$H \leftarrow B_1, \ldots, B_k,$$

where $H, B_1, \ldots, B_k \in \mathsf{HB}_P$. The case $k = 0$ is a fact and is written simply as $H \leftarrow$.

To match Section 4.1, write $P = \mathcal{B} \cup \mathcal{P}$, where $\mathcal{B}$ contains the ground facts and $\mathcal{P}$ contains rules with nonempty bodies. We identify each fact $A \leftarrow$ with its head A. All ground instances of empty-body clauses are included in $\mathcal{B}$ before initialization.

For a non-ground program P, let $\mathsf{ground}(P)$ denote the set of all ground instances of clauses of P. Since Herbrand-model semantics is defined over ground atoms, the algorithm below is stated for $\mathsf{ground}(P)$. Thus, unless explicitly noted otherwise, all clauses are assumed ground.

A Herbrand interpretation is a set $I \subseteq \mathsf{HB}_P$. It satisfies a ground clause

$$H \leftarrow B_1, \ldots, B_k$$

when

$$\{B_1, \ldots, B_k\} \subseteq I \quad \Longrightarrow \quad H \in I.$$

The immediate consequence operator of P is the monotone map

$$T_P : \mathcal{P}(\mathsf{HB}_P) \to \mathcal{P}(\mathsf{HB}_P)$$

defined by

$$T_P(I) = \{\, H \in \mathsf{HB}_P : \text{there is } H \leftarrow B_1, \ldots, B_k \in \mathsf{ground}(P) \text{ with } \{B_1, \ldots, B_k\} \subseteq I \,\}.$$

The conventional least Herbrand model of P is

$$\mathsf{LM}(P) = \mathrm{lfp}(T_P) = \bigcup_{n<\omega} T_P^n(\varnothing).$$

A.2 SHIFTED UNBOUND VALUATIONS

In the shifted unbound semantics, truth values are real numbers and truth is thresholded at 0. We use the following abstraction.

Definition A.1 (Shifted valuation)**.** A shifted valuation over HB_P is a function

$$v : \mathsf{HB}_P \to \mathbb{R}.$$

The Boolean projection of v is

$$\tau(v) = \{\, A \in \mathsf{HB}_P : v(A) \geq 0 \,\}.$$

Atoms in $\tau(v)$ are called active or true, and atoms outside $\tau(v)$ are called inactive or false.

For body atoms $B_1, \ldots, B_k$, the shifted additive conjunction would assign the value

$$v(B_1 \& \cdots \& B_k) = \sum_{i=1}^{k} v(B_i).$$

This operation does not preserve falsity. For example,

$$v(B_1) = -1, \qquad v(B_2) = 2$$

gives

$$v(B_1 \& B_2) = 1 \geq 0,$$

even though B_1 is false. Therefore additive conjunction cannot be used as the rule-applicability test for Horn forward chaining.

Instead, rule applicability is checked using a gate.

Definition A.2 (Gate and payload)**.** Let

$$r = H \leftarrow B_1, \ldots, B_k$$

be a ground definite Horn clause and let v be a shifted valuation.

The gate of r under v is open when

$$\mathsf{Gate}_r(v) \quad \Longleftrightarrow \quad \min_{1 \leq i \leq k} v(B_i) \geq 0.$$

For $k = 0$, the minimum condition is vacuously true.

The payload of r under v is

$$\mathsf{Pay}_r(v) = \sum_{i=1}^{k} v(B_i),$$

with the empty sum defined as 0.

Thus the gate determines whether the rule fires, while the payload determines the value sent to the head.

Lemma A.3 (Gate correctness)**.** *For every shifted valuation v and every ground rule*

$$r = H \leftarrow B_1, \ldots, B_k,$$

we have

$$\mathsf{Gate}_r(v) \quad \Longleftrightarrow \quad \{B_1, \ldots, B_k\} \subseteq \tau(v).$$

Proof. By definition,

$$\mathsf{Gate}_r(v) \quad \Longleftrightarrow \quad \min_i v(B_i) \geq 0.$$

This holds exactly when $v(B_i) \geq 0$ for every i, which is exactly the condition $B_i \in \tau(v)$ for every i. □

A.3 The gated additive consequence operator

Throughout the proof, assume that $\mathsf{ground}(P)$ is finite. This includes propositional Horn programs and function-free programs over a finite domain, as considered in this paper.

As in Section 4.1, initialize

$$v_0(A) = \begin{cases} 1, & A \in \mathcal{B}, \\ -1, & A \notin \mathcal{B}. \end{cases}$$

Define the total incoming contribution at stage n by

$$D_n(A) = \sum_{c \in \mathcal{P}} \sum_{\substack{r \in \mathsf{ground}(c) \\ \mathsf{head}(r) = A}} \mathbf{1}[\mathsf{Gate}_r(v_n)]\, \mathsf{Pay}_r(v_n).$$

The synchronous update is

$$v_{n+1}(A) = v_n(A) + D_n(A). \tag{9}$$

This is exactly Eq. 5: every gate and payload is evaluated using v_n, and all fired-rule contributions are added to the previous head value. An empty collection of contributions sums to 0. Facts are seeded at 1 rather than derived through an empty-body payload, whose value would be 0.

Algorithm 1 LuHorn forward chaining (Eq. 5)

Require: Ground background facts $\mathcal{B}$; definite Horn rules $\mathcal{P}$ with nonempty bodies; finite Herbrand base HB of $\mathcal{B} \cup \mathcal{P}$
Ensure: The least Herbrand model $\mathsf{LM}(\mathcal{B} \cup \mathcal{P})$

for all $a \in \mathrm{HB}$ **do**
  $v^{(0)}(a) \leftarrow 1$ if $a \in \mathcal{B}$, and -1 otherwise
end for
$n \leftarrow 0$
repeat
  $v^{(n+1)} \leftarrow v^{(n)}$ {Copy the current valuation}
  $\mathsf{closed} \leftarrow \mathsf{true}$
  for all $c \in \mathcal{P}$ and $r \in \mathrm{ground}(c)$ **do**
    if $\min_{b \in \mathrm{body}(r)} v^{(n)}(b) \geq 0$ **then**
      $a \leftarrow \mathrm{head}(r)$
      $v^{(n+1)}(a) \leftarrow v^{(n+1)}(a) + \sum_{b \in \mathrm{body}(r)} v^{(n)}(b)$
      if $v^{(n)}(a) < 0$ **then**
        $\mathsf{closed} \leftarrow \mathsf{false}$
      end if
    end if
  end for
  $n \leftarrow n + 1$
until $\mathsf{closed} = \mathsf{true}$
return $\{a \in \mathrm{HB} : v^{(n)}(a) \geq 0\}$

Algorithm 1 implements Eq. 5 with synchronous updates: all gates and body sums use $v^{(n)}$, while contributions from all ground rules accumulate in $v^{(n+1)}$. All facts, including empty-body clauses, are included in $\mathcal{B}$. The algorithm stops when the true atoms are closed under the ground rules, rather than requiring the unbounded numerical valuations to converge. In particular, an enabled rule with an inactive head keeps $\mathsf{closed} = \mathsf{false}$ even if its current payload is zero. The following results establish why this stopping condition is necessary and why the algorithm terminates with the correct model.

A.4 Correspondence with classical forward chaining

For each n, let

$$S_n = \tau(v_n) = \{A \in \mathsf{HB}_P : v_n(A) \geq 0\}.$$

Lemma A.4 (Threshold invariant)**.** *For every $n < \omega$ and every $A \in \mathsf{HB}_P$,*

$$v_n(A) \in \{-1\} \cup \mathbb{Z}_{\geq 0}, \qquad v_{n+1}(A) \geq v_n(A).$$

The contributions are nonnegative integers and satisfy

$$D_{n+1}(A) \geq D_n(A) \geq 0.$$

Consequently, $S_n \subseteq S_{n+1}$, and an inactive atom has value exactly -1.

Proof. The initial values are 1 or -1. If the values at stage n are integers, each open gate contributes a sum of nonnegative integers, and each closed gate contributes zero. Thus $D_n(A)$ is a nonnegative integer. By Eq. 9, valuations remain integers and never decrease. Since their initial values are at least -1, they lie in $\{-1\} \cup \mathbb{Z}_{\geq 0}$.

An open gate remains open as the valuations increase, and its payload cannot decrease. A newly opened gate contributes a nonnegative payload. Hence each rule's contribution, and therefore $D_n(A)$, is nondecreasing in n. □

Lemma A.5 (Positive values after activation)**.** *If $A \in S_n$, then $v_{n+1}(A) \geq 1$.*

Proof. If $v_n(A) \geq 1$, the claim follows from monotonicity. Otherwise $v_n(A) = 0$. Since no atom is initialized at zero, there is a first stage j, with $1 \leq j \leq n$, at which $v_j(A) = 0$. The threshold invariant gives $v_{j-1}(A) = -1$, so $D_{j-1}(A) = 1$. Since the contributions are nondecreasing, $D_n(A) \geq 1$. Therefore $v_{n+1}(A) = v_n(A) + D_n(A) \geq 1$. □

Proposition A.6 (Correspondence with at most two updates)**.** *For every $n < \omega$,*

$$S_n \subseteq S_{n+1} \subseteq S_n \cup T_P(S_n) \subseteq S_{n+2}.$$

Proof. The first inclusion follows from Lemma A.4. If $A \in S_{n+1} \setminus S_n$, then $v_n(A) = -1$ and $D_n(A) \geq 1$. Thus at least one rule with head A has an open gate. By Lemma A.3, its body is contained in S_n, giving $A \in T_P(S_n)$. This proves the second inclusion.

For the last inclusion, atoms in S_n remain active. Consider $A \in T_P(S_n) \setminus S_n$. Since every fact is already in S_0, there is a rule $A \leftarrow B_1, \ldots, B_k$ with $k \geq 1$ and all $B_i \in S_n$. By Lemma A.5, $v_{n+1}(B_i) \geq 1$ for every i. At stage $n+1$, this rule therefore contributes at least $k \geq 1$ to A. Since $v_{n+1}(A) \geq -1$, we obtain $v_{n+2}(A) \geq 0$. □

In general, $S_{n+1} = S_n \cup T_P(S_n)$ need not hold. For example, let $\mathcal{B} = \{p\}$ and $\mathcal{P} = \{q \leftarrow p,\ r \leftarrow q\}$. In the order (p, q, r), the valuations are

$$v_0 = (1, -1, -1), \quad v_1 = (1, 0, -1), \quad v_2 = (1, 1, -1), \quad v_3 = (1, 2, 0).$$

The rule $r \leftarrow q$ is enabled at stage 1, but its zero payload leaves r inactive at stage 2. Thus $S_1 = S_2$ even though these sets are not yet closed under P. This is why the algorithm checks rule closure rather than stopping after a single unchanged Boolean projection.

A.5 Soundness and completeness

We now prove that the gated additive procedure computes exactly the conventional least Herbrand model.

Theorem A.7 (Soundness)**.** *For every $n < \omega$,*

$$S_n \subseteq \mathsf{LM}(P).$$

Consequently, every atom returned by Algorithm 1 belongs to the conventional least Herbrand model of P.

Proof. We prove the claim by induction on n.

For $n = 0$, we have $S_0 = \mathcal{B} \subseteq \mathsf{LM}(P)$, since every Herbrand model of P contains its facts.

Assume $S_n \subseteq \mathsf{LM}(P)$. Since $\mathsf{LM}(P)$ is a Herbrand model of P, it is closed under T_P:

$$T_P(\mathsf{LM}(P)) \subseteq \mathsf{LM}(P).$$

By monotonicity of T_P,

$$T_P(S_n) \subseteq T_P(\mathsf{LM}(P)) \subseteq \mathsf{LM}(P).$$

Using Proposition A.6,

$$S_{n+1} \subseteq S_n \cup T_P(S_n) \subseteq \mathsf{LM}(P).$$

Therefore $S_n \subseteq \mathsf{LM}(P)$ for every n. □

Theorem A.8 (Completeness)**.** *Let*

$$S_\omega = \bigcup_{n<\omega} S_n.$$

Then

$$\mathsf{LM}(P) \subseteq S_\omega.$$

If HB_P is finite, Algorithm 1 terminates and returns a set containing every atom of $\mathsf{LM}(P)$.

Proof. We show that S_ω is a Herbrand model of P. It contains all facts because $S_0 = \mathcal{B}$.

Let

$$H \leftarrow B_1, \ldots, B_k$$

be a ground rule of $\mathcal{P}$ with $k \geq 1$, and suppose

$$B_1, \ldots, B_k \in S_\omega.$$

Since the body is finite, for each i there exists $n_i < \omega$ such that $B_i \in S_{n_i}$. Let

$$m = \max\{n_1, \ldots, n_k\}.$$

The sequence $(S_n)_{n<\omega}$ is monotone by Proposition A.6, so

$$B_1, \ldots, B_k \in S_m.$$

Therefore

$$H \in T_P(S_m).$$

Again by Proposition A.6,

$$H \in S_{m+2} \subseteq S_\omega.$$

Thus S_ω satisfies every ground clause of P, so S_ω is a Herbrand model.

Since $\mathsf{LM}(P)$ is the least Herbrand model,

$$\mathsf{LM}(P) \subseteq S_\omega.$$

If HB_P is finite, every failure of closure at stage n gives an atom in $T_P(S_n) \setminus S_n$. By Proposition A.6, this atom enters S_{n+2}. Thus closure cannot keep failing without adding new atoms within two updates, and only finitely many atoms can be added. The algorithm therefore reaches a stage n at which $T_P(S_n) \subseteq S_n$ and terminates. At that stage, the same proposition gives $S_{n+1} = S_n$, and induction shows that all subsequent Boolean projections equal S_n. Hence the returned set is S_ω. □

Corollary A.9 (Correctness)**.** *For every definite Horn program P with finite grounding and the initialization and updates above,*

$$S_\omega = \mathsf{LM}(P).$$

In particular, when HB_P is finite, Algorithm 1 terminates and returns exactly the conventional least Herbrand model of P.

Proof. By Theorem A.7,

$$S_\omega \subseteq \mathsf{LM}(P).$$

By Theorem A.8,

$$\mathsf{LM}(P) \subseteq S_\omega.$$

Therefore $S_\omega = \mathsf{LM}(P)$. □

A.6 Discussion

The correctness argument depends on the distinction between the gate and the payload. If one used the shifted additive conjunction itself as the body test, then a rule could fire even when some body atom is false, because negative and positive body values may cancel. The gate prevents this by testing

$$\min_i v(B_i) \geq 0,$$

which is exactly the classical Horn condition that every body atom has already been derived. Once the gate is open, the rule contributes its additive payload to the head's accumulated value. Summing all such contributions gives

$$v_{n+1}(H) = v_n(H) + D_n(H).$$

The specified $\{-1, 1\}$ initialization ensures that a true atom with value zero becomes strictly positive after one more update. Consequently, an enabled rule derives its head within at most two updates, and the union of the Boolean projections is exactly the ordinary least Herbrand model.

B Rule extraction

After training, we extract atoms from each parameterized rule using a dynamic Top-K heuristic (Algorithm 2). We sort the atoms by descending absolute weight and retain those preceding the first position where the ratio of the current weight magnitude to the previous one falls below a threshold $\tau \in (0, 1)$, thereby determining K adaptively. If no such cutoff occurs, all atoms are retained; if all weights are zero, no atoms are selected.

Algorithm 2 Dynamic Top-K Rule Extraction

Input: Feature set $\mathcal{F}$ and trained weights $\{|w(f)|\}_{f\in\mathcal{F}}$ for one rule; ratio threshold $\tau \in (0, 1)$
Output: Selected atom set $\mathcal{S}$

$n \leftarrow |\mathcal{F}|$
$s_i \leftarrow |w(f_i)|$ for $f_i \in \mathcal{F}$
Sort indices $\pi_1, \ldots, \pi_n$ such that $s_{\pi_1} \geq \cdots \geq s_{\pi_n}$
if $s_{\pi_1} = 0$ **then**
  return $\emptyset$
end if
$K \leftarrow n$
for $j = 2, \ldots, n$ **do**
  $r_j \leftarrow s_{\pi_j}/s_{\pi_{j-1}}$
  if $r_j < \tau$ **then**
    $K \leftarrow j - 1$
    break
  end if
end for
$\mathcal{S} \leftarrow \{f_{\pi_j} \mid 1 \leq j \leq K\}$
return $\mathcal{S}$

C Implementation details for visual ILP experiments

Hardware and seeds. αILP and the neural baselines run on NVIDIA A100-40GB GPUs. NEUMANN runs on NVIDIA V100-SXM2-32GB GPUs with CUDA 11.8, PyTorch 2.1.2, PyTorch-Geometric 2.5.3 and Python 3.10. All experiments are repeated over five random seeds. Reported numbers are mean $\pm$ standard deviation.

C.1 Dataset details

Image transforms. For Kandinsky, both released code-bases load images via the same data class `data_kandinsky.KANDINSKY`, which resizes the source 640×640 RGBA images to 128×128 using `cv2.resize` (default `img_size=128`), converts BGR→RGB and HWC→CHW, and forwards the tensor directly to the pre-trained YOLOv5 detector. No ImageNet mean/std normalisation is applied at this stage. For CLEVR-Hans3, both αILP and NEUMANN follow the Slot-Attention default, `transforms.Resize((128, 128))` together with the standard ImageNet normalisation.

C.2 Full hyper-parameter listing

Neural baselines: architecture. All architectures and losses below were verified against the official αILP and NeSy-XIL repositories. **CNN – Kandinsky.** A torchvision `resnet18` (`pretrained=False`, $\sim$ 11.7 M parameters) outputs a 1000-dim feature, on top of which a logistic regression head (`Linear`$(1000, 1)$ followed by sigmoid) produces the binary prediction. The whole stack is jointly trained with Adam and binary cross-entropy loss (`train_cnn.py`). **CNN – CLEVR-Hans3.** The `ResNet34Small` of, namely `torchvision.resnet34` initialised with ImageNet weights and truncated to drop `layer4`, `avgpool` and `fc`, followed by an `AdaptiveAvgPool2d`$(1, 1)$ and `Linear`$(256, 3)$. The network performs 3-way classification with `CrossEntropyLoss` and Adam (`NeSyXIL/clevr_hans/cnn/train_resnet34.py`). **YOLO+MLP.** The pre-trained `YOLOv5s` detector (`train=False`, 7.09 M frozen parameters, `YOLOPerceptionModule(e=6, d=11)`) emits a fixed 6×11 object tensor that is flattened to 66 dimensions, on top of which a 3-layer MLP (`Linear`$(66, 256) \rightarrow$ ReLU $\rightarrow$ `Linear`$(256, 256) \rightarrow$ ReLU $\rightarrow$ `Linear`$(256, 1) \rightarrow$ sigmoid, hidden size 256) produces the binary prediction. Adam and binary cross-entropy loss (`train_yolo_mlp.py`). **NeSy (SA+MLP).** The pre-trained Slot-Attention perception module (`SlotAttentionPerceptionModule(e=10, d=19)`) outputs a 10×19 object tensor flattened to 190 dimensions, on top of which the same 3-layer MLP head ($190 \rightarrow 256 \rightarrow 256 \rightarrow 1$, ReLU, sigmoid) maps to the binary label. Adam and

binary cross-entropy loss (`train_sa_mlp_clevr.py`). Note that this is the SA+MLP variant shipped in the αILP repository.

Table 4: NEUMANN per-task ILP hyper-parameters as used in our runs (`baselines/neumann/run.py`).

Task	N_{trial}	N_{sample}	program size	max-var	min-body-len	infer-step	num-objects
twopairs	5	20	5	6	3	4	6
closeby	5	20	5	6	3	4	6
red-triangle	4	20	5	6	3	4	6
CLEVR-Hans3/0	5	10	1	6	3	2	10
CLEVR-Hans3/1	6	10	1	6	3	2	10
CLEVR-Hans3/2	6	10	1	6	3	2	10

D LIMITATIONS AND FUTURE WORK

Possible global gradient issues Our work focuses on alleviating the local gradient issues caused by fuzzy logic operators. Although the unbound logic LuHorn alleviates those local gradient issues, global gradients may also be problematic, which is leave to future works.

Soundness and completeness of DiffLu Although LuHorn (Eq. 5) is proved to be sound and complete, its continuously relaxation DiffLu (Eq. 7) may be not well-formed due to the relaxation of gates. This is common in continuous relaxation of discrete structures. Many continuous relaxations suffer from interpolated results that are not interpretable in the discrete sense, which may cause optimization issues. In the future, we will further improve the soundness and completeness of DiffLu.

Failure analysis Although DiffLu outperforms the strongest directly comparable baseline by a large margin on average, it performs poor on Cyclic and Len tasks. In addition to the possible failure reasons discussed above, we have the following observation after inspecting the reference solutions provided by Evans & Grefenstette (2018).

- Cyclic: This appears to be the only task that must rely on the recursion rule for auxiliary predicates. For other tasks that require auxiliary predicates but do not require recursion on them, we can simply drop the need for auxiliary predicates, which results in a longer rule (i.e., with more body atoms). Regarding our failed experiment, our initial guess is that our model is currently not robust to unavoidable auxiliary predicates.
- Len: This appears to be the only task that must rely on the predicates containing the same variables. We currently do not consider this case, as shown in the example feature set of Section 4.2.

Consider that DiffLu demonstrates great performance on other recursive reasoning tasks, including LessThan, Member, Connectedness, EvenSucc, EvenOdd, Buzz, and Fizz, we leave Cyclic and Len to future works.

Time complexity The time complexity of our method is almost the same as classical forward chaining.

- From Eq. 7, the time complexity is linear wrt. the number of rules and reasoning steps.
- From Eq. 7 and the definition of feature set $\mathcal{F}$, the time complexity is linear wrt. the number of predicates since $|\mathcal{F}|$ is.
- Another crutial factor is the number of constants $|D|$. Typically, a straightforward substitution algorithm examines roughly $|D|^V$, where V is the number of variables in a rule. In substitution, there should be an algorithm to check facts. A naive scan will incur an additional factor of the number of facts.

Reducing time complexity is a meaningful future work.

Weaker interpretability of vDiffLu We have proved that DiffLu can truly learn logical rules from symbolic data, and it is also effective on visual ILP problems. However, due to the lack of training on intermediate annotations, vDiffLu's interpretability on visual ILP tasks is not as good as αILP and NEUMANN. Although we can extract logical rules from vDiffLu, the meaning of the predicates and constants is not clear. We do find interesting interpretations of vDiffLu on the red triangle task. The perception model nearly always outputs a slot that attends to the red triangles in the images. In the future, we will explore how to recover the same level of interpretability with αILP and NEUMANN.