Machine Learning (2024) 113

Learning Differentiable Logic Programs for Abstract Visual Reasoning

Hikaru Shindo · Viktor Pfanschilling · Devendra Singh Dhami · Kristian Kersting

Accepted: Aug 4, 2024

Abstract Visual reasoning is essential for building intelligent agents that understand the world and perform problem-solving beyond perception. Differentiable forward reasoning has been developed to integrate reasoning with gradient-based machine learning paradigms. However, due to the memory intensity, most existing approaches do not bring the best of the expressivity of first-order logic, excluding a crucial ability to solve *abstract visual reasoning*, where agents need to perform reasoning by using analogies on abstract concepts in different scenarios. To overcome this problem, we propose *NEUro-symbolic Message-pAssiNg reasoNer (NEUMANN)*, which is a graph-based differentiable forward reasoner, passing messages in a memory-efficient manner and handling structured programs with functors. Moreover, we propose a computationally-efficient structure learning algorithm to perform explanatory program induction on complex visual scenes. To evaluate, in addition to conventional visual reasoning tasks, we propose a new task, *visual reasoning behind-the-scenes*, where agents need to learn abstract programs and then answer queries by imagining scenes that are not observed. We empirically demonstrate that NEUMANN solves visual reasoning tasks efficiently, outperforming neural, symbolic, and neuro-symbolic baselines.

1 Introduction

Deep Neural Networks (DNNs) are attracting considerable interest due to significant performances in crucial tasks in Artificial Intelligence [51] such as image recognition [46], game playing [92], protein-structure prediction [39], and language modeling [11] to name a few. DNNs are essentially data-driven, *i.e.* they perform pattern recognition statistically given data and perform prediction on new examples. However, a critical gap exists between human intel-

Hikaru Shindo
E-mail: hikaru.shindo@tu-darmstadt.de

ligence and the current data-driven machine-learning paradigm. Humans can explain and understand what they see, imagine things they could see but have not yet, and perform planning to solve problems [47]. Moreover, humans can learn from a small number of experiences [99,93], but DNNs such as transformers [101,28,109,27,108] require a large dataset to achieve good performance on a specific task [32]. These essential intelligent aspects of humans, called *model building* [47], are vital for human-level intelligence.

Logic has been a fundamental element of AI for providing knowledge representations and reasoning capabilities [22,85]. Inductive Logic Programming (ILP) [63,71,16] is a framework to learn logic programs given examples. In stark contrast to DNNs, ILP gains some crucial advantages, *e.g.* it can learn from small data, and it can learn explicit programs, which are interpretable by humans. Recently, Differentiable ILP (∂ILP) has been proposed [29], where they perform gradient-based learning of logic programs. In ∂ILP, *forward reasoning*, which derives all possible consequences given logic programs, is implemented using only differentiable operations by encoding logic programs into *tensors*. Thus it can be easily combined with DNNs for the perception and perform ILP on visual inputs. However, tensor-based differentiable forward reasoning is memory-intensive. Thus it assumes that logic programs to be handled are simple, *e.g.* each predicate takes at most two arguments, each clause has at most two body atoms, and no functors are allowed. ∂ILP-ST [90] has been developed to deal with structured logic programs with functors in ∂ILP, leading to αILP [91], which can learn classification rules on complex visual scenes. They address the memory-consumption problem by performing a beam-search over clauses instead of generating all possible clauses by templates. However, performing a beam search is computationally expensive because every candidate of clauses needs to be evaluated in each step. Thus it takes longer to complete when handling complex programs and does not scale for more challenging tasks where agents play multiple roles, *e.g.* understanding visual scenes, learning abstract operations and solving queries by abstract reasoning.

To mitigate this issue, we develop a memory-efficient differentiable forward reasoner and a computationally efficient learning strategy. We propose *NEUro-symbolic Message-pAssiNg reasoNer (NEUMANN)*, a graph-based approach for differentiable forward reasoning, sending messages in a memory-efficient manner. We first introduce a new graph-based representation of logic programs in first-order logic and then perform differentiable reasoning via message passing. The graph structure efficiently encodes the reasoning process by connecting logical atoms. Then, we propose a computationally-efficient learning algorithm for NEUMANN by combining gradient-based scoring and differentiable sampling. Instead of scoring each clause exactly to perform a beam search, NEUMANN computes gradients over candidate clauses for a classification loss and uses them as approximated scores to generate new clauses. By doing so, NEUMANN avoids nested scoring loops over clauses, which has been a computational bottleneck of the beam-search approach.

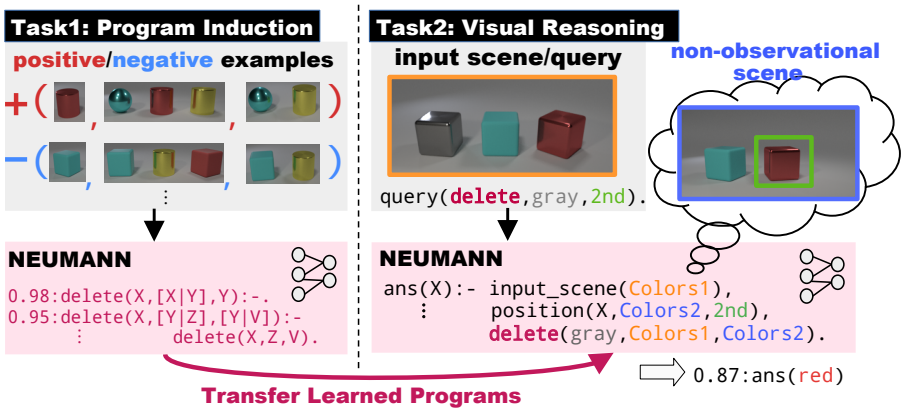


Fig. 1 Reasoning behind the scenes: The goal of this task is to compute the answer to a query, *e.g. "What is the color of the second left-most object after deleting a gray object?"* given a visual scene. To answer this query, the agent needs to reason behind the scenes and understand abstract operations on objects. **(Task 1, left)** In the first task, the agent needs to induce an explicit program given visual examples, where each example consists of several visual scenes that describe the input and the output of the operation to be learned. The abstract operations can be described by first-order logic with functors. **(Task 2, right)** In the second task, the agent needs to apply the learned programs to new situations to solve queries about non-observational scenes. (Best viewed in color)

The memory-efficient reasoning and computationally-efficient learning enable NEUMANN to solve *abstract visual reasoning*, where the agent needs to perform reasoning by using analogies on abstract concepts in different scenarios. To evaluate this, we propose a new task, *Visual Reasoning Behind the Scenes*, where the agent needs to perform complex visual reasoning imagining scenes that are not observed. Fig. 1 illustrates a Behind-the-Scenes task whose goal is to compute the answer of a query, *e.g. "What is the color of the second left-most object after deleting a gray object?"* given a visual scene. In turn, it consists of two sub-tasks. The first is to induce abstract programs from visual scenes, *e.g. deletion* of objects, as shown in the left of Fig. 1. The second is to solve the queries where the answers are derived by reasoning about non-observational scenes. To solve, the agent needs to learn abstract operations from visual input and perform efficient reasoning. The task assesses the following four essential model-building capacities: (1) learning from a small number of examples, (2) understanding complex visual scenes deeply, (3) learning explanatory programs to transfer to new tasks, and (4) imagining situations that have not been observed directly. Behind-the-Scenes is the first benchmark to cover all of these four aspects. We highlight on Tab. 1 the difference from previous visual reasoning tasks in these aspects. Behind-the-Scenes serves as a legitimate task and dataset for the model-building abilities, which is beneficial to foster the machine-learning paradigm to perform problem-solving beyond pattern recognition.

To summarize, we make the following important contributions:

Table 1 Comparison between Behind-the-Scenes and other visual reasoning benchmarks. The Behind-the-scenes task assesses the four essential *model-building* features: **(small data)** the task requires the model to learn from a small number of training data, **(visual scenes)** the task requires to handle complex visual scenes where several objects appear, **(explanatory)** the task requires to learn explanatory programs, and **(imagination)** the task requires answers obtained by reasoning about non-observational scenes.

	small data	visual scenes	explanatory	"imagination"
VQA [2]	✗	✓	✗	✗
VQAR [36]	✗	✓	✓	✗
CLEVR [38]	✗	✓	✗	✗
CLEVRER [106]	✗	✓	✗	✓
CLEVR-Hans [97]	✗	✓	✓	✗
MNIST-Addition [58]	✗	✗	✓	✗
RAVEN [77]	✓	✓	✗	✗
KandinskyPattern [66]	✓	✓	✓	✗
Behind-the-Scenes	✓	✓	✓	✓

1. We propose NEUMANN[1], a memory-efficient differentiable forward reasoner using message-passing. We theoretically and empirically show that NEUMANN requires less memory than conventional tensor-based differentiable forward reasoners [29,90,91]. Given G ground atoms and C^* ground clauses, conventional differentiable forward reasoners consume memory quadratically $\mathcal{O}(G \times C^*)$, but NEUMANN consumes linearly $\mathcal{O}(G + C^*)$.
2. We propose a computationally-efficient learning algorithm for NEUMANN to learn complex programs from visual scenes. NEUMANN performs gradient-based scoring and differentiable sampling, avoiding nested loops for scoring candidate clauses.
3. We propose a new challenging task and a dataset, *Visual Reasoning Behind the Scenes*, where the agents need to perform abstract visual learning and reasoning on complex visual scenes. The task requires the agents to learn abstract operations from small data on visual scenes and reason about non-observational scenes to answer queries. The task evaluates machine-learning models on the different essential model-building properties of intelligence beyond perception, which are not covered by the previously addressed visual reasoning benchmarks.
4. We empirically show that NEUMANN solves visual reasoning tasks such as Kandinsky patterns [66] and CLEVR-Hans [97] using less memory than conventional differentiable forward reasoners, outperforming neural baselines. More importantly, we show that NEUMANN efficiently solves the proposed Behind-the-Scenes task, outperforming conventional differentiable forward reasoners. To this end, we show that NEUMANN gains the advantages of scalable and explainable visual reasoning and learning against symbolic and neuro-symbolic baselines.

[1] Code is available: `https://github.com/ml-research/neumann`

2 First-Order Logic, Differentiable Reasoning, and Graph Neural Networks

Before introducing NEUMANN, we revisit the basic concepts of first-order logic and graph neural networks.

First-Order Logic (FOL). A *Language* $\mathcal{L}$ is a tuple $(\mathcal{P}, \mathcal{A}, \mathcal{F}, \mathcal{V})$, where $\mathcal{P}$ is a set of predicates, $\mathcal{A}$ is a set of constants, $\mathcal{F}$ is a set of function symbols (functors), and $\mathcal{V}$ is a set of variables. A *term* is a constant, a variable, or a term that consists of a functor. A *ground term* is a term with no variables. We denote n-ary predicate $\mathtt{p}$ by $\mathtt{p}/n$. An *atom* is a formula $\mathtt{p}(\mathtt{t_1}, \ldots, \mathtt{t_n})$, where $\mathtt{p}$ is an n-ary predicate symbol and $\mathtt{t_1}, \ldots, \mathtt{t_n}$ are terms. A *ground atom* or simply a *fact* is an atom with no variables. A *literal* is an atom or its negation. A *positive literal* is just an atom. A *negative literal* is the negation of an atom. A *clause* is a finite disjunction ($\vee$) of literals. A *ground clause* is a clause with no variables. A *definite clause* is a clause with exactly one positive literal. If $A, B_1, \ldots, B_n$ are atoms, then $A \vee \neg B_1 \vee \ldots \vee \neg B_n$ is a definite clause. We write definite clauses in the form of A :- $B_1, \ldots, B_n$. Atom A is called the *head*, and set of negative atoms $\{B_1, \ldots, B_n\}$ is called the *body*. We call definite clauses by clauses for simplicity in this paper. We denote *true* as $\top$ and *false* as $\bot$. Substitution $\theta = \{\mathtt{X_1} = \mathtt{t_1}, ..., \mathtt{X_n} = \mathtt{t_n}\}$ is an assignment of term $\mathtt{t_i}$ to variable $\mathtt{X_i}$. An application of substitution θ to atom A is written as $A\theta$. An atom is an atomic *formula*. For formula F and G, $\neg F$, $F \wedge G$, and $F \vee G$ are also formulas. *Interpretation* of language $\mathcal{L}$ is a tuple $(\mathcal{D}, \mathcal{I}_{\mathcal{A}}, \mathcal{I}_{\mathcal{F}}, \mathcal{I}_{\mathcal{P}})$, where $\mathcal{D}$ is the domain, $\mathcal{I}_{\mathcal{A}}$ is the assignments of an element in $\mathcal{D}$ for each constant $\mathtt{a} \in \mathcal{A}$, $\mathcal{I}_{\mathcal{F}}$ is the assignments of a function from $\mathcal{D}^n$ to $\mathcal{D}$ for each n-ary function symbol $\mathtt{f} \in \mathcal{F}$, and $\mathcal{I}_{\mathcal{P}}$ is the assignments of a function from $\mathcal{D}^n$ to $\{\top, \bot\}$ for each n-ary predicate $\mathtt{p} \in \mathcal{P}$. For language $\mathcal{L}$ and formala X, an interpretation $\mathcal{I}$ is a *model* if the truth value of X w.r.t $\mathcal{I}$ is true. Formula X is a *logical consequence* or *logical entailment* of a set of formulas $\mathcal{H}$, denoted $\mathcal{H} \models X$, if, $\mathcal{I}$ is a model for $\mathcal{H}$ implies that $\mathcal{I}$ is a model for X for every interpretation $\mathcal{I}$ of $\mathcal{L}$.

(Differentiable) Forward Reasoning is a data-driven approach of reasoning in FOL [85]. Forward reasoning is performed by applying a function called the $T_{\mathcal{C}}$ *operator*, deducing new ground atoms using given clauses and ground atoms. For a set of clauses $\mathcal{C}$, $T_{\mathcal{C}}$ operator [54] is a function that applies clauses in $\mathcal{C}$ using given ground atoms $\mathcal{G}$, *i.e.*

$$T_{\mathcal{C}}(\mathcal{G}) = \mathcal{G} \cup \left\{ A \;\middle|\; \begin{array}{l} A \text{ :- } B_1, \ldots, B_n \in \mathcal{C}^* \\ (\{B_1 \ldots, B_n\} \subseteq \mathcal{G}) \end{array} \right\}, \tag{1}$$

where $\mathcal{C}^*$ is a set of all ground clauses that can be produced from $\mathcal{C}$. Note that the union with $\mathcal{G}$ is computed to hold the ground atoms in the previous steps. The forward reasoning function can then be defined as a function that repeatedly applies the $T_{\mathcal{C}}$ operator to given ground atoms.

Differentiable forward reasoning [29,90,91] uses only simple tensor operations to compute forward reasoning. Given G ground atoms and C clauses,

the reasoner computes the grounding of clauses, *i.e.* removing variables, producing C^* ground clauses, then it builds *index tensor* $\mathbf{I} \in \mathbb{N}^{G \times C^*}$, which holds the indices of ground atoms for each ground clause. The differentiable forward reasoner computes logical entailment referring to the index tensor repeatedly.

Graph Neural Networks. Graph Neural Network (GNN) [87,53,43,33, 88] is a type of neural network that processes graphs as inputs. An input data is represented as $(\mathsf{G}, \mathbf{x}_{node}, \mathbf{x}_{edge})$, where G is a directed or undirected graph, $\mathbf{x}_{node}$ represents node features, and $\mathbf{x}_{edge}$ represents edge features. Given an input, GNN computes the node representations by performing message-passing:

$$x_i^{(t+1)} = f_{update}\left(x_i^{(t)}, \bigoplus\nolimits_{j \in \mathcal{N}(i)} c_{ji} \cdot x_j^{(t)}\right), \tag{2}$$

where $t \in \mathbb{N}$ is a time step, $x_i^{(t)}$ is a node feature of node $\mathsf{x_i}$ at time step t, $\mathcal{N}(i)$ is a set of indices of neighbors of node $\mathsf{x_i}$, c_{ji} is an edge feature, and $\bigoplus$ is an aggregation function to aggregate messages from neighbors.

3 NEUMANN

NEUMANN computes logical entailment in a differentiable manner given visual input and weighted clauses. Fig. 2 illustrates the overview of NEUMANN's reasoning pipeline. In contrast to conventional differentiable forward reasoners [29,90,91], NEUMANN performs message-passing on graphs in the following steps: **(Step 1)** A visual input is fed into a neural network to perceive objects in the scene. The output of the neural network is encoded into a set of probabilistic atoms $\mathbf{x}_{atom}^{(0)}$. **(Step 2)** Given input probabilistic atoms $\mathbf{x}_{atom}^{(0)}$, NEUMANN performs T bi-directional message-passing steps. The graph represents a set of weighted clauses, and the output node features $\mathbf{x}_{atom}^{(T)}$ represent probabilistic values of logical entailment given $\mathbf{x}_{atom}^{(0)}$ and weighted clauses. We describe each step in detail.

3.1 Forward Reasoning Graph

We represent a set of weighted clauses as a directed bipartite graph. Fig. 3 shows an example of a set of weighted clauses and a corresponding forward reasoning graph. Intuitively, the graph has two groups of nodes representing nodes of ground atoms and nodes of conjunctions. Edges represent how the ground clauses connect the ground atoms and conjunctions with their weights.

Definition 1 A *Forward Reasoning Graph* is a bipartite directed graph $(\mathcal{V}_\mathcal{G}, \mathcal{V}_\wedge, \mathcal{E}_{\mathcal{G}\to\wedge}, \mathcal{E}_{\wedge\to\mathcal{G}})$, where $\mathcal{V}_\mathcal{G}$ is a set of nodes representing ground atoms (atom nodes), $\mathcal{V}_\wedge$ is set of nodes representing conjunctions (conjunction nodes), $\mathcal{E}_{\mathcal{G}\to\wedge}$ is set of edges from atom to conjunction nodes and $\mathcal{E}_{\wedge\to\mathcal{G}}$ is a set of edges from conjunction to atom nodes.

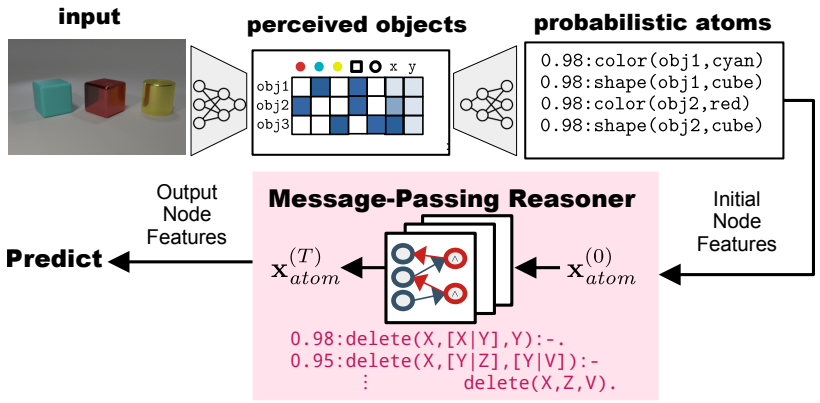


Fig. 2 The reasoning architecture in NEUMANN. Raw input images are fed into the visual-perception model. The output is converted into a set of probabilistic ground atoms. Differentiable forward reasoning is performed by a bi-directional message-passing algorithm. Logical entailment is computed softly using weighted clauses and probabilistic ground atoms. (Best viewed in color)

Algorithm 1 Building a forward reasoning graph

Input: clauses $\mathcal{C}$, ground atoms $\mathcal{G}$, language $\mathcal{L}$
1: RG $\leftarrow$ `init`($\mathcal{G}$) # add $\top$ node and atom nodes for $\mathcal{G}$
2: $\mathcal{C}^*$ $\leftarrow$ `ground_clauses`($\mathcal{C}, \mathcal{L}$) # compute all possible groundings for clauses $\mathcal{C}$
3: **for** $C_i^* = A$:- $B_1, \ldots, B_n \in \mathcal{C}^*$ **do**
4: add conjunction node $\mathsf{C_i^*}$ to RG
5: **for** $B_j \in [B_1, \ldots, B_n]$ **do**
6: add edge $(\mathsf{B_j}, \mathsf{C_i^*})$ to RG # Atom2Conj edge
7: **end for**
8: add edge $(\mathsf{C_i^*}, \mathsf{A})$ to RG # Conj2Atom edge
9: **end for**
Output: RG

Given a set of clauses and ground atoms, Algorithm 1 shows the construction of a corresponding forward reasoning graph. **(Line 1)** First, the graph is initialized by adding the atom nodes for ground atoms $\mathcal{G}$ and a special node $\top$, which represents *true*. It is used to represent clauses that have no body atoms, *e.g.* `r(X):-`. **(Line 2)** The function `ground_clauses` takes a set of clauses and a language as input. In general, an infinite number of ground terms can be considered with functors in FOL. Thus we consider a subset of the ground terms by limiting the number of nested functors. A set of ground clauses $\mathcal{C}^*$ is obtained by substituting ground terms for variables. **(Line 3–8)** For each ground clause, $C_i^* \in \mathcal{C}^*$, corresponding node and edges are added to the reasoning graph, *i.e.* edges from body atoms to a conjunction, and from a conjunction to a head atom. X denotes the corresponding node in the graph for a logical formula X. Each ground clause corresponds to a conjunction node in the reasoning graph.

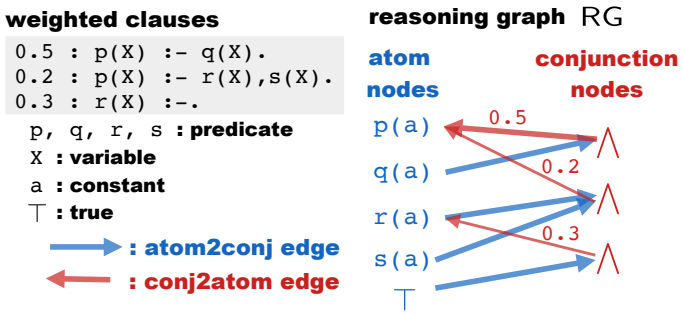


Fig. 3 Example Forward Reasoning Graph. Weighted clauses (left) and a corresponding reasoning graph (right). Blue nodes represent ground atoms, and red nodes represent conjunctions. Each conjunction node corresponds to each ground clause. Edges represent how the ground clauses connect the ground atoms and conjunctions. (Best viewed in color)

3.2 Message Passing for Forward Chaining

NEUMANN performs forward-chaining reasoning by passing messages on the reasoning graph. Essentially, forward reasoning consists of *two* steps: (1) computing conjunctions of body atoms for each clause and (2) computing disjunctions for head atoms deduced by different clauses. These two steps can be efficiently computed on bi-directional message-passing on the forward reasoning graph. We now describe each step in detail.

(Direction $\rightarrow$) From Atom to Conjunction. First, messages are passed to the conjunction nodes from atom nodes. For conjunction node $\mathsf{v_i} \in \mathcal{V}_\wedge$, the node features are updated:

$$v_i^{(t+1)} = \bigvee \left(v_i^{(t)}, \bigwedge\nolimits_{j \in \mathcal{N}(i)} v_j^{(t)} \right), \tag{3}$$

where $\bigwedge$ is a soft implementation of *conjunction*, and $\bigvee$ is a soft implementation of *disjunction*. Intuitively, probabilistic truth values for bodies of all ground clauses are computed softly by Eq. 3.

(Direction $\leftarrow$) From Conjunction to Atom. Following the first message passing, the atom nodes are then updated using the messages from conjunction nodes. For atom node $\mathsf{v_i} \in \mathcal{V}_\mathcal{G}$, the node features are updated:

$$v_i^{(t+1)} = \bigvee \left(v_i^{(t)}, \bigvee\nolimits_{j \in \mathcal{N}(i)} w_{ji} \cdot v_j^{(t)} \right), \tag{4}$$

where w_{ji} is a weight of edge $e_{j \rightarrow i}$. We assume that each clause $C_k \in \mathcal{C}$ has its weight θ_k, and $w_{ji} = \theta_k$ if edge $e_{j \rightarrow i}$ on the reasoning graph is produced by clause C_k. Intuitively, in Eq. 4, new atoms are deduced by gathering values from different ground clauses and from the previous step.

Performing message-passing by Eq. 3-4 corresponds to deducing new atoms by Eq. 1 in FOL using probabilistic inputs and weighted clauses. We used

Algorithm 2 Reasoning on NEUMANN

Input: input scene s, reasoning graph RG, clause weights $\mathbf{w}$, background knowledge $\mathcal{B}$, reasoning step T, target atom G_i

1: $\mathbf{x}_{atoms}^{(0)} = f_{perceive}(s, \mathcal{B})$ # visual perception
2: **for** $t \in [1, \ldots, T]$ **do**
3: # massages from atom nodes to conjunction nodes
4: $\mathbf{x}_{conj}^{(t)} = atom2conj(\mathbf{x}_{atoms}^{(t-1)}, \mathsf{RG})$
5: # massages from conjunction nodes to atom nodes using clause weights
6: $\mathbf{x}_{atoms}^{(t)} = conj2atom(\mathbf{x}_{conj}^{(t)}, \mathsf{RG}, \mathbf{w})$
7: **end for**
8: # extract the value of the target atom G_i
9: $p(G_i \mid \mathbf{x}_{atoms}^{(0)}, \mathsf{RG}, \mathbf{w}, \mathcal{B}, T) = \mathbf{x}_{atoms}^{(T)}[i]$

Output: $p(G_i \mid \mathbf{x}_{atoms}^{(0)}, \mathsf{RG}, \mathbf{w}, \mathcal{B}, T)$

product for conjunction, and *log-sum-exp* function [18] for disjunction:

$$softor^{\gamma}(x_1, \ldots, x_n) = \gamma \log \sum_{1 \leq i \leq n} \exp(x_i/\gamma), \tag{5}$$

where $\gamma > 0$ is a smooth parameter. Eq. 5 approximates the maximum value given input $x_1, \ldots, x_n$ in a differentiable manner.

3.3 Prediction

The probabilistic logical entailment is computed by the bi-directional message-passing. Let $\mathbf{x}_{atoms}^{(0)} \in [0,1]^{|\mathcal{G}|}$ be input node features, which map a ground atom to a scalar value, RG be the reasoning graph, $\mathbf{w}$ be the clause weights, $\mathcal{B}$ be background knowledge, and $T \in \mathbb{N}$ be the infer step. For ground atom $G_i \in \mathcal{G}$, NEUMANN computes the probability as follows:

$$p(G_i \mid \mathbf{x}_{atoms}^{(0)}, \mathsf{RG}, \mathbf{w}, \mathcal{B}, T) = \mathbf{x}_{atoms}^{(T)}[i], \tag{6}$$

where $\mathbf{x}_{atoms}^{(T)} \in [0,1]^{|\mathcal{G}|}$ is the node features of atom nodes after T-steps of the bi-directional message-passing.

Algorithm 2 summarizes the reasoning steps on NEUMANN. **(Line 1)** Input scene s is converted to probabilistic atoms $\mathbf{x}_{atoms}^{(0)}$ by the perception function $f_{perceive}$. We used the perception module of αILP [91], which performs visual perception to produce object-centric representations, and converts them to probabilistic atoms. Given background knowledge is also incorporated to produce $\mathbf{x}_{atoms}^{(0)}$. **(Line 2-4)** For each reasoning time step, the messages are propagated from the atom nodes to the conjunction nodes by Eq. 3. **(Line 5-6)** The messages are propagated from the conjunction nodes to the atom nodes by Eq. 4. **(Line 8-9)** The value for the target atom G_i is extracted by Eq. 6 and returned.

3.4 NEUMANN Memory Consumption

We now compare NEUMANN to conventional differentiable forward reasoners [29,90,91]. NEUMANN achieves memory-efficient reasoning by message-passing.

Proposition 1 *Let $\mathcal{G}$ be a set of ground atoms and $\mathcal{C}$ be a set of clauses, which produce a set of ground clauses $\mathcal{C}^*$ with a language $\mathcal{L}$. The memory consumption of the reasoning graph is $\mathcal{O}\left(|\mathcal{G}| + |\mathcal{C}^*|\right)$, while that of the conventional differentiable forward-chaining tensors is $\mathcal{O}\left(|\mathcal{G}| \times |\mathcal{C}^*|\right)$.*

Proof. The number of atom nodes is $|\mathcal{G}|$, and the number of the conjunction nodes is $|\mathcal{C}^*|$. Thus, the memory consumption by the nodes is $\mathcal{O}(|\mathcal{G}| + |\mathcal{C}^*|)$. For each ground clause $C^* = A$:- $B_1, \ldots, B_n \in \mathcal{C}^*$, each body atom B_i is connected to a conjunction node, *i.e.* n edges, and another edge from the conjunction node to a head atom A. Thus, the memory consumption of the edges is $\mathcal{O}(|\mathcal{C}^*| \times (n+1))$. To this end, the total memory consumption of the sum of those of the nodes and edges, *i.e.* $\mathcal{O}(|\mathcal{G}| + |\mathcal{C}^*| + |\mathcal{C}^*|(n+1)) \approx \mathcal{O}(|\mathcal{G}| + |\mathcal{C}^*|)$. The tensor-based reasoners build a tensor $\mathbf{I} \in \mathbb{N}^{|\mathcal{G}| \times |\mathcal{C}^*|}$, which holds the indices of ground atoms for each ground clause. Thus the overall memory consumption is $\mathcal{O}(|\mathcal{G}| \times |\mathcal{C}^*|)$.

3.5 Learning Logic Programs by NEUMANN

Now we describe how NEUMANN searches logic programs given a visual ILP problem.

Problem Statement. Let $\mathcal{Q} = (\mathcal{E}^+, \mathcal{E}^-, \mathcal{B}, \mathcal{L}, \mathcal{Z})$ be a visual ILP problem, where $\mathcal{E}^+$ is a set of positive examples, $\mathcal{E}^-$ is a set of negative examples, $\mathcal{B}$ is background knowledge, $\mathcal{L}$ is a language, and $\mathcal{Z}$ is a language bias. Each example is given as a visual scene. The task is to find a logic program that can perform classification correctly based on the attributes and relations of objects in the scenes.

Fig. 4 shows an overview of learning of NEUMANN. It learns logic programs in *two* steps: (1) NEUMANN generates promising clauses by iterating scoring and sampling of clauses. Candidate clauses are evaluated by computing their gradients for a classification loss, and promising clauses are sampled via differentiable sampling using the Gumbel-max trick. To this end, new candidate clauses are generated by refining the sampled clauses, and a new reasoning graph is produced. (2) After the iteration of clause generation steps, NEUMANN assigns randomly-initialized clause weights and optimizes them to minimize the classification loss. It uses stochastic gradient descent for the optimization.

We first describe the clause-generation step and weight-optimization step in detail, respectively, and then we explain the whole learning algorithm for NEUMANN, highlighting the difference from existing differentiable ILP solvers.

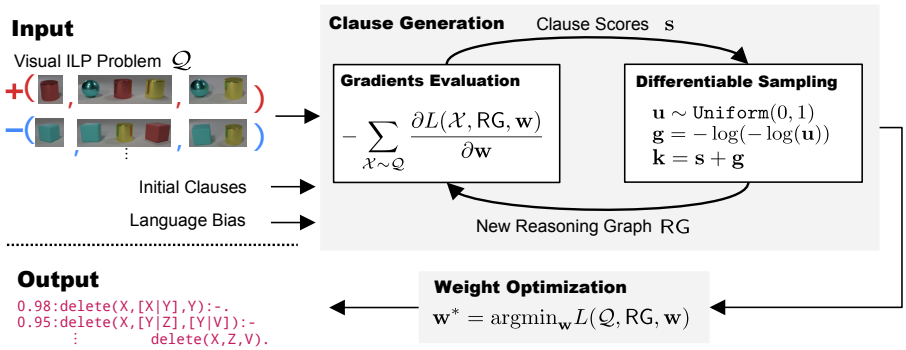


Fig. 4 Structure learning on NEUMANN. Given positive and negative examples as visual scenes, NEUMANN learns logic programs by the following *two* steps. (1) NEUMANN generates promising clauses by iterating scoring and sampling of clauses. Candidate clauses are evaluated by computing their gradients for a classification loss, and promising clauses are sampled via differentiable sampling. To this end, new candidate clauses are generated by refining the sampled clauses, and a new reasoning graph is produced. (2) After the iteration of clause generation steps, NEUMANN assigns randomly-initialized clause weights and optimizes them to minimize the classification loss. (Best viewed in color)

3.5.1 Clause Generation

NEUMANN generates candidate clauses by iteratively (1) scoring clauses using gradients and (2) performing differentiable sampling on the scores and refining them. We extend the beam search approach used in αILP [91] to achieve more efficient clause generation using gradients avoiding nested loops.

Clause Scoring by Gradients. NEUMANN generates candidates of clauses $\mathcal{C}$ by refining given initial clauses $\mathcal{C}_0$ repeatedly. We evaluate clauses by computing gradients at once. By using the end-to-end reasoning architecture, NEUMANN scores each clause efficiently using automatic differentiation. Given a visual ILP problem $\mathcal{Q}$, a reasoning graph RG, clause weights $\mathbf{w}$, and background knowledge $\mathcal{B}$, NEUMANN computes the binary-cross entropy loss:

$$L(\mathcal{Q}, \mathsf{RG}, \mathbf{w}) = -\mathbb{E}_{(e,y)\sim\mathcal{Q}}[y \log p(y \mid e, \mathsf{RG}, \mathbf{w}, \mathcal{B}, T) + (1-y)\log(1 - p(y \mid e, \mathsf{RG}, \mathbf{w}, \mathcal{B}, T))], \tag{7}$$

where (e, y) is a tuple of a visual scene e and its label y, *i.e.* if e is a positive example then $y = 1$ otherwise $y = 0$. The conditional probability of the label $p(y \mid e, \mathsf{RG}, \mathbf{w}, \mathcal{B}, T)$ is computed by using Eq. 6.

Using the loss, NEUMANN scores candidate clauses $\mathcal{C}$ by computing gradients w.r.t. the clause weights, *i.e.* NEUMANN computes clause scores $\mathbf{s} \in \mathbb{R}^{|\mathcal{C}|}$:

$$\mathbf{s} = -\sum_{\mathcal{X}\sim\mathcal{Q}} \frac{\partial L(\mathcal{X}, \mathsf{RG}, \mathbf{w})}{\partial \mathbf{w}}, \tag{8}$$

where $\mathcal{X}$ is a sampled batch of labeled examples, RG is a reasoning graph constructed using clauses $\mathcal{C}$, and $\mathbf{w} \in \mathbb{R}^{|\mathcal{C}|}$ is a clause weight. Intuitively, useful

clauses to classify given visual scenes get negatively large gradients to minimize the classification loss. Thus we compute the negative raw gradients and consider them as the evaluation scores, *i.e.* promising clauses that contribute much to classify examples correctly will get high scores. For scoring, all clauses are associated with a uniform value to exclude the influence of the difference in weight values. Note that we do not update the clause weights $\mathbf{w}$ in this step but compute gradients to score clauses.

Example. Suppose we want to solve a simple classification of visual scenes with a pattern: *"If there is a red cube, the scene is positive."*, *e.g.* a scene in Fig. 2 is a positive example. The task is to learn a classification rule in FOL:

$$\texttt{positive(X):-in(O1,X),color(O1,red),shape(O1,cube).}$$

We start from a general clause $\texttt{positive(X):-in(O1,X).}$, and by refining, we get, *e.g.*

$$\texttt{positive(X):-in(O1,X),color(O1,red).}$$
$$\texttt{positive(X):-in(O1,X),color(O1,blue).}$$
$$\texttt{positive(X):-in(O1,X),color(O1,yellow).}$$

We compose a reasoning graph using these three clauses and give a uniform weight to all clauses. Using the reasoning graph, we compute the scores by Eq. 8. The first clause contributes the most to correct classifications and thus is scored higher than other clauses. NEUMANN performs inference over given visual scenes only once to score all clauses, not iterating it for each individual clause.

Fig. 5 illustrates the difference from the conventional clause-scoring strategy. The task is to score candidate clauses $\mathcal{C}$ given visual ILP problem $\mathcal{Q}$ to perform clause search. In ∂ILP-ST [90] and αILP [91], each clause $C_i \in \mathcal{C}$ needs to be evaluated individually, and thus the computational cost increases quadratically with respect to the number of training data and the number of clauses to be evaluated. In contrast, NEUMANN evaluates all clauses by calling the backward function once.

Generation by Differentiable Sampling. Given clause scores $\mathbf{s}$, we generate new candidates of clauses by performing differentiable sampling based on the Gumbel-max trick [37,56] and refine them. The Gumbel-max simulates efficiently sampling procedures given scores in a differentiable manner. For the clause scores $\mathbf{s}$, a noise term is computed as $\mathbf{g} = -\log(-\log(\mathbf{u}))$ where $\mathbf{u} \sim \texttt{Uniform}(0, 1)$. Then we add the noise to the original scores as $\mathbf{k} = \mathbf{s} + \mathbf{g}$, *i.e.* $\mathbf{k}$ represents scores mixed with a Gumbel noise. Then clause $C_i \in \mathcal{C}$ is sampled with $i = argmax(k_1, \ldots, k_{|\mathcal{C}|})$. The sampled clauses are refined using *downward refinement operator* [71], which specifies given clauses, *i.e.* generates more specific clauses than the given clause in terms of the number of atoms to be entailed with it. Given clause C (*e.g.* $\texttt{p(X,Y):-.}$), the refinement operator consists of the following *four* specifications: (i) add an atom to the body of C (*e.g.* $\texttt{p(X,Y):-q(X,Y).}$), (ii) substitute a constant to variable in C (*e.g.* $\texttt{p(X,a):-.}$), (iii) remove a variable by substituting another variable in C

Scoring in αILP / ∂ILP-ST

```
# initialize clause scores
s = 0
# for each batch of examples
for X ~ Q
  # for each clause to be scored
  for C_i ∈ C
    # scoring by computing loss
    s[i] += − L(X, RG, w)
```

Scoring in NEUMANN

```
# initialize clause scores
s = 0
# for each batch of examples
for X ~ Q
  # score all clauses by gradients
  s += − ∇_w L(X, RG, w)
```

Fig. 5 NEUMANN avoids nested loops for clause scoring. Input is a visual ILP problem $\mathcal{Q}$ and clauses $\mathcal{C}$, output is the scores $\mathbf{s}$ over $\mathcal{C}$. αILP [91] and ∂ILP-ST [90] evaluate each clause independently. In contrast, NEUMANN evaluates a set of clauses efficiently without performing for-loop over them. $\mathcal{X} \sim \mathcal{Q}$ denotes a sampled batch of examples from visual ILP problem $\mathcal{Q}$.

($\mathtt{p(X,X):-.}$), and (iv) apply a functor (*e.g.* $\mathtt{p(X,f(Y,Z)):-.}$). Downward refinement operator ensures *completness*, *i.e.* any clauses that consist of a finite set of symbols can be generated by applying the operator for finite times to the most general clause [71]. NEUMANN uses mode declarations [64] (*cf.* App. D) to restrict the search space, and clauses that do not satisfy the declarations will be discarded. The newly generated clauses are added to the set of clauses $\mathcal{C}$ and evaluate the added clauses by Eq. 8 in the next step.

3.5.2 Weight Optimization

After the clause generation, NEUMANN performs loss minimization for classification with respect to clause weights. So far, we assumed that we have one clause weight vector. By using softmax, NEUMANN can learn to select one clause out of multiple generated clauses. However, in practice, we should be able to learn logic programs consisting of multiple clauses.

NEUMANN composes differentiable logic programs that consist of multiple clauses as follows: (1) We fix the target programs' size as M, *i.e.* where we try to find a logic program with M clauses out of generated clauses $\mathcal{C}$. (2) We introduce randomly-initialized $|\mathcal{C}|$-dimensional weights $\mathbf{W} = [\mathbf{w}^{(1)}, \ldots, \mathbf{w}^{(M)}]$ ($\mathbf{w}^{(j)} \in \mathbb{R}^{|\mathcal{C}|}$), *i.e.* each clause gets M individual weights. (3) We take softmax of each weight vector $\mathbf{w}^{(j)} \in \mathbf{W}$ and softly choose M clauses out of $|\mathcal{C}|$ clauses, *i.e.* $\hat{\mathbf{w}}^{(j)} = \mathit{softmax}(w_0^{(j)}, \ldots, w_{|\mathcal{C}|}^{(j)})$. (4) We compose a clause weight vector $\mathbf{w} \in [0,1]^{|\mathcal{C}|}$ as:

$$w_i = \mathit{softor}^{\gamma}\left(\hat{w}_i^{(1)}, \ldots, \hat{w}_i^{(M)}\right) = \gamma \log \sum_{1 \le j \le M} \exp(\hat{w}_i^{(j)}/\gamma), \tag{9}$$

where $\gamma > 0$ is a smooth parameter, approximating the maximum value out of M weights for each clause in a differentiable manner.

For example, suppose 3 clauses are generated by NEUMANN, and we want to compose a logic program that consists of 2 clauses, *i.e.* $|\mathcal{C}| = 3$ and $M = 2$. By initializing 2 weight vectors and applying softmax to each, we

get, *e.g.* $\hat{\mathbf{w}}^{(1)} = [0.1, 0.7, 0.2]^\top$ and $\hat{\mathbf{w}}^{(2)} = [0.8, 0.1, 0.1]^\top$. Using Eq. 9, we get $\mathbf{w} \approx [0.8, 0.7, 0.2]^\top$, where the first 2 clauses get large weights.

The weights for the clauses are trained to minimize the loss function. By using the end-to-end reasoning architecture, NEUMANN finds a logic program that explains the complex visual scenes by gradient descent, *i.e.* solves

$$\mathbf{w}^* = \text{argmin}_{\mathbf{w}} L(\mathcal{Q}, \mathsf{RG}, \mathbf{w}), \tag{10}$$

where L is the cross-entropy loss (Eq. 7). NEUMANN minimizes the loss based on stochastic gradient descent. After performing sufficient weight-update steps, the generated clauses and their trained weights are returned.

Algorithm 3 shows the entire learning process of NEUMANN. **(Line 1-3)** An initial reasoning graph is built. **(Line 5-10)** Clauses $\mathcal{C}$ are scored by computing gradients. Useful clauses in $\mathcal{C}$ get negatively large gradients, and thus they are scored high at line 10. **(Line 13-21)** Sample clauses to be refined to generate new clauses according to the scores using the Gumbel-max trick. **(Line 22-25)** The sampled clauses are refined to generate clauses to be scored in the next iteration. **(Line 27-32)** NEUMANN performs weight optimization using the generated clauses $\mathcal{C}_{sampled}$ with randomly initialized clause weights $\mathbf{w}$.

We highlight the difference between NEUMANN and other differentiable ILP approaches in terms of the memory cost and clause-search cost in Tab. 2. As shown in Prop. 1, NEUMANN consumes less memory than other approaches, *i.e.* NEUMANN consumes memory linearly with the number of ground atoms and clauses, but others consume quadratically. ∂ILP generates clauses by templates without any symbolic search. Thus it requires no cost for searching but needs to exclude functors to manage the number of clauses to be generated. ∂ILP-ST and αILP perform beam search using exact scoring of clauses. As illustrated in Fig. 5, the time complexity of exact scoring is $\mathcal{O}(N_{data} \times |\mathcal{C}| \times R)$, where N_{data} is the number of data, $\mathcal{C}$ is the set of clauses to be scored, and R is the time complexity of the reasoning function. Although they require nested loops for data and clauses, they can handle functors because beam search can prune redundant clauses. In contrast, NEUMANN computes forward and backward pass for each data to evaluate clauses $\mathcal{C}$, and thus the time complexity of the scoring is $\mathcal{O}(N_{data} \times (R + R)) \approx \mathcal{O}(N_{data} \times R)$ because both forward and backward pass have the time complexity of R. The scoring of clauses needs to be conducted at every step of the search, and thus it is crucial to have an efficient scoring strategy.

4 Experiments

We empirically show that NEUMANN is a memory-efficient differentiable forward reasoner equipped with a computationally-efficient learning algorithm by solving visual reasoning tasks. Moreover, we show that NEUMANN solves the proposed *Behind-the-Scenes* task, where different model-building abilities are required beyond perception. To this end, we show that NEUMANN can

Algorithm 3 Learning NEUMANN

Input: visual ILP problem $\mathcal{Q}$, ground atoms $\mathcal{G}$, language $\mathcal{L}$, initial clauses $\mathcal{C}_0$, language bias $\mathcal{Z}$, search parameters N_{trial}, N_{samlpe}, target program size M

$\mathsf{RG} = \texttt{build\_reasoning\_graph}(\mathcal{C}_0, \mathcal{G}, \mathcal{L})$ # initialize a reasoning graph
$\mathcal{C}_{sampled} = \phi$ # all sampled clauses
$\mathcal{C} = \mathcal{C}_0$ # clauses to be scored next
# perform clause-generation for N_{trial} times
for $n \in [1, \ldots, N_{trial}]$ **do**
    # clause evaluation by computing gradients
    $\mathbf{s} = \mathbf{0}$ # initialize clause scores
    for $\mathcal{X} \sim \mathcal{Q}$ **do**
        # compute scores by gradients with coefficient $\beta > 0$
        $\mathbf{s} = \mathbf{s} + \beta \cdot (-\nabla_{\mathbf{w}} L(\mathcal{X}, \mathsf{RG}, \mathbf{w}))$
    end for
    # sample N_{sample} clauses based on the scores
    $\mathcal{D}_{sampled} = \phi$ # sampled clauses at step n
    for $m \in [1, \ldots, N_{sample}]$ **do**
        # sample N_{sample} clauses using the Gumbel-max trick
        $\mathbf{u} \sim \texttt{Uniform}(0, 1)$
        $\mathbf{g} = -\log(-\log(\mathbf{u}))$
        $\mathbf{k} = \mathbf{s} + \mathbf{g}$
        $C_i \in \mathcal{C}$ is sampled with $i = argmax(k_1, \ldots, k_{|\mathcal{C}|})$
        add C_i to $\mathcal{D}_{sampled}$
    end for
    # generate new clauses to be scored in the next iteration using language bias
    $\mathcal{C} = downward\_refinement(\mathcal{D}_{sampled}, \mathcal{L}, \mathcal{Z})$
    # update all sampled clauses
    $\mathcal{C}_{sampled} = \mathcal{C}_{sampled} \cup \mathcal{D}_{sampled}$
end for
# initialize a reasoning graph using all sampled clauses
$\mathsf{RG} = \texttt{build\_reasoning\_graph}(\mathcal{C}_{sampled}, \mathcal{G}, \mathcal{L})$
initialize the clause weights according to the target program size M
$\mathbf{w} = \texttt{initialize\_weights}(\mathcal{C}_{sampled}, M)$
clause weight optimization by stochastic gradient descent
$\mathbf{w}^* = \text{argmin}_{\mathbf{w}} L(\mathcal{Q}, \mathsf{RG}, \mathbf{w})$

Output: $\mathcal{C}_{sampled}, \mathbf{w}^*$

perform scalable visual reasoning and learning and provide visual explanations efficiently, outperforming existing symbolic and neuro-symbolic benchmarks. We implemented NEUMANN using PyTorch. All experiments were performed on one NVIDIA A100-SXM4-40GB GPU with Xeon(R):8174 CPU@3.10GHz and 100 GB of RAM.

We aim to answer the following questions:

Q1: Does the message-passing reasoning algorithm simulate the differentiable forward reasoning dealing with uncertainty?
Q2: Can NEUMANN solve visual ILP problems combined with DNNs outperforming neural baselines and consuming less memory than the other differentiable ILP benchmarks?
Q3: Does NEUMANN solve the Behind-the-Scenes task outperforming conventional differentiable reasoners providing the model-building abilities (*cf.* Tab. 1)?

Table 2 NEUMANN is a memory-efficient differentiable ILP solver equipped with an efficient learning algorithm. A comparison of memory consumption, search cost for each step, scoring method, and the capability of handling functors with other differentiable ILP solvers. $\mathcal{G}$ is a set of ground atoms, $\mathcal{C}$ is a set of clauses, and $\mathcal{C}^*$ is a set of ground clauses. N_{data} is the number of examples, R is the time complexity of the differentiable forward chaining.

	Memory Cost	Search Cost per Step	Scoring Method	Functors
∂ILP [29]	$\mathcal{O}(\lvert\mathcal{G}\rvert \times \lvert\mathcal{C}^*\rvert)$	$\mathcal{O}(1)$	No Scoring (Template)	✗
∂ILP-ST [90]	$\mathcal{O}(\lvert\mathcal{G}\rvert \times \lvert\mathcal{C}^*\rvert)$	$\mathcal{O}(N_{data} \times \lvert\mathcal{C}\rvert \times R)$	Exact Scoring	✓
αILP [91]	$\mathcal{O}(\lvert\mathcal{G}\rvert \times \lvert\mathcal{C}^*\rvert)$	$\mathcal{O}(N_{data} \times \lvert\mathcal{C}\rvert \times R)$	Exact Scoring	✓
NEUMANN	$\mathcal{O}(\lvert\mathcal{G}\rvert + \lvert\mathcal{C}^*\rvert)$	$\mathcal{O}(N_{data} \times R)$	Gradient-based	✓

Q4: Does NEUMANN provide advantages over state-of-the-art symbolic and neuro-symbolic methods?

4.1 Differentiable Reasoning with Uncertainty

To answer **Q1**, we compare NEUMANN with a conventional tensor-based differentiable forward reasoner, αILP [91], and show that both reasoners produce almost the same proof histories dealing with uncertainties given the same input. We explore two datasets used in ∂ILP [29].

Even/Odd. Even/Odd is a synthetic dataset to classify even and odd numbers. We used the following program:

$$1.0 : \texttt{even}(\texttt{s}^2(\texttt{X})) \texttt{:-} \texttt{even}(\texttt{X}).$$

and background knowledge $\mathcal{B} = \{\texttt{even}(\texttt{0})\}$. $\texttt{s}$ is a functor that represents *sucessor* of natural numbers, *e.g.* natural number 2 can be represented by a term $\texttt{s}(\texttt{s}(\texttt{0}))$. The task is to deduce even numbers given the rule about even numbers and the base fact that 0 is an even number.

Fig. 6 shows the proof history produced by NEUMANN and αILP for the even/odd dataset. Each element of the x-axis represents a ground atom. For the y-axis, from top to bottom, each row represents a vector of probabilities over atoms for 5-steps of differentiable forward reasoning, *i.e.* $\mathbf{x}^{(0)}_{atoms}, \ldots, \mathbf{x}^{(5)}_{atoms}$. Both reasoners deduced step by step the following atoms, $\texttt{even}(\texttt{0}), \texttt{even}(\texttt{s}^2(\texttt{0}))$, $\ldots, \texttt{even}(\texttt{s}^{10}(\texttt{0}))$, with high probabilities, which are almost 1.0, but not any odd numbers, *i.e.* they successfully deduced even numbers producing almost the same proof histories. The message-passing algorithm simulates forward reasoning (Eq. 1) in FOL.

Cyclic Graph. Cyclic Graph is a dataset to classify if each node in a graph is cyclic or not. We used the same directed graph used in ∂ILP [29] as shown in Fig. 7 (top). We used the following weighted clauses to describe the rules of cyclicity:

$$0.51 : \texttt{cyclic}(\texttt{X}) \texttt{:-} \texttt{edge}(\texttt{X}, \texttt{X}).$$
$$0.54 : \texttt{edge}(\texttt{X}, \texttt{Y}) \texttt{:-} \texttt{edge}(\texttt{X}, \texttt{Z}), \texttt{edge}(\texttt{Z}, \texttt{Y}).$$

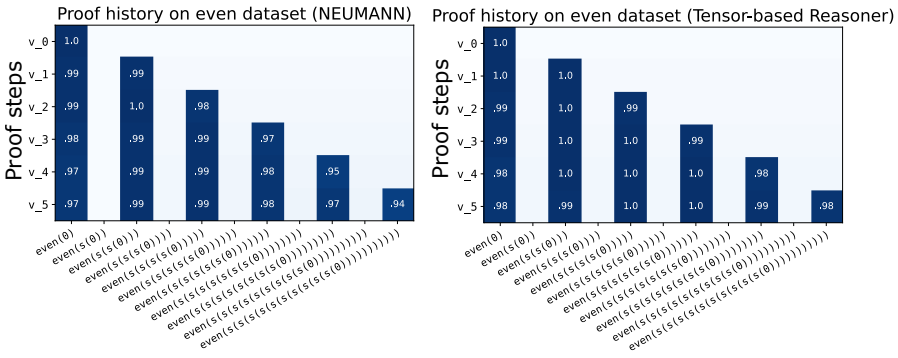


Fig. 6 NEUMANN performs differentiable forward reasoning. Proof histories in the Even/Odd task [29] by NEUMANN (left) and αILP [91]. NEUMANN produces almost the same values as the tensor-based reasoner.

and background knowledge to represent the graph:

$$\mathcal{B} = \left\{ \begin{array}{l} \texttt{edge}(\texttt{a},\texttt{b}),\ \texttt{edge}(\texttt{b},\texttt{c}),\ \texttt{edge}(\texttt{b},\texttt{d}),\ \texttt{edge}(\texttt{c},\texttt{a}), \\ \texttt{edge}(\texttt{d},\texttt{e}),\ \texttt{edge}(\texttt{d},\texttt{f}),\ \texttt{edge}(\texttt{e},\texttt{f}),\ \texttt{edge}(\texttt{f},\texttt{e}) \end{array} \right\}.$$

$\texttt{cyclic}(\texttt{X})$ means node $\texttt{X}$ is a cyclic node, *i.e.* there is a path to trace that starts and ends at node $\texttt{X}$. The task is to deduce whether each node is a cyclic node or not, given the set of nodes and weighted clauses.

Fig. 7 shows a proof history produced by NEUMANN and αILP for the Cyclic Graph dataset. We show the proof history of the 5-steps of differentiable forward reasoning. Both reasoners produced almost the same probabilities for each ground atom at each time step. More importantly, both reasoners deal with uncertainty, *i.e.* since the given programs have different weights and thus reasoners need to compute probabilistic values for each ground atom according to the weights, and NEUMANN successfully simulates the differentiable forward reasoning by the message-passing algorithm. These results show that the message-passing reasoning on NEUMANN is a valid differentiable forward reasoning function, even though it consumes much less memory than the tensor-based differentiable forward reasoners, *e.g.* ∂ILP, ∂ILP-ST, and αILP.

4.2 Differentiable ILP on Complex Visual Scenes

To answer **Q2**, we compare the performance of NEUMANN with conventional differentiable ILP solvers and neural baselines on visual reasoning tasks and show obtained explanatory programs. We also compare the memory consumption of NEUMANN with conventional differentiable forward reasoners.

Dataset. We adopted Kandinsky patterns [66] and CLEVR-Hans [97] dataset. Both datasets are defined as a classification task of visual scenes, and the classification rules are defined by attributes of the objects and their

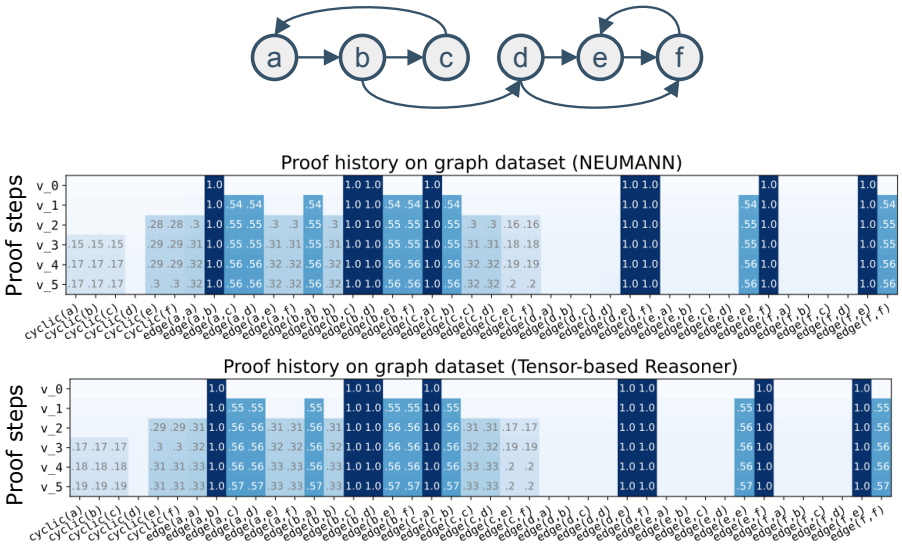


Fig. 7 NEUMANN performs differentiable forward reasoning. A directed graph (top) and proof histories in the Cyclic Graph task [29] by NEUMANN and αILP [91] (bottom). NEUMANN produces almost the same values as the tensor-based reasoner dealing with uncertainty.

relations. Fig. 8 shows examples of the patterns we used. We use 5 Kandinsky patterns: *(P1) twopairs*, *(P2) closeby*, *(P3) red-triangle*, *(P4) online-pair*, and *(P5) long-line*. (P5) is an extension of (P4) where the number of objects is increased to 7 and the constraints of pairing are removed. We performed structure learning on (P1)-(P4), and for (P5), we used a given clause[2] to perform reasoning to assess the scalability of logic reasoners for many objects. The dataset contains 10k training examples for each pattern for each positive and negative class, respectively. Likewise, each validation and test split contains 5k examples for each positive and negative class. The CLEVR-Hans dataset [97] contains CLEVR [38] 3D images, and each image is associated with a class label. Examples for each are shown in Fig. 8. We consider 3 binary classification tasks for each pattern. Each class contains 3k training images, 750 validation images, and 750 test images, respectively. More examples for both datasets are available in App. E.

Models. For Kandinsky patterns, we compare NEUMANN against *two* neural baselines and a differentiable ILP baseline. We adopted the ResNet-based CNN model [34] as a benchmark and also an object-centric benchmark, YOLO+MLP, where the input figure is fed to the pre-trained YOLO model [79]. The output of the pre-trained YOLO model is fed into MLP with 2 hidden layers with nonlinearity to predict the class label. We trained the whole YOLO+MLP network jointly. For CLEVR-Hans tasks, the considered baselines are the ResNet-based CNN model [34], and the Neuro-Symbolic (NeSy)

[2] $\texttt{kp5(X)} : -\texttt{in(O1, X)}, \texttt{in(O2, X)}, \texttt{in(O3, X)}, \texttt{in(O4, X)}, \texttt{in(O5, X)}, \texttt{in(O6, X)}, \texttt{in(O7, X)}, \texttt{online(O1, O2, O3, O4, O5, O6, O7)}.$

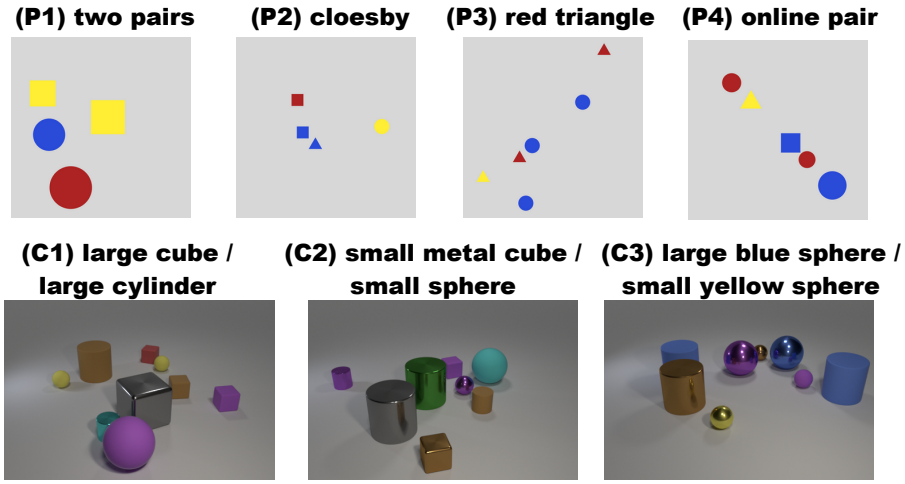


Fig. 8 Examples of Kandinsky patterns and CLEVR-Hans datasets. The task is to classify visual scenes by obtaining explicit classification rules. Each Kandinsky pattern is characterized as follows: (**twopairs**) *A pair with the same shape and color, and another pair with the same shape and different colors.* (**closeby**) *Two objects closely located.* (**red-triangle**) *A red triangle close to a non-triangle and non-red object.* (**online-pair**) *Five objects aligned on a line and a pair with the same shape and color.* (**long-line**) *Seven objects aligned on a line.* (Best viewed in color)

model [97]. The NeSy model uses slot attention [55] to perceive objects and feeds its output to Set Transformer [52]. NeSy-XIL is a NeSy model trained using additional supervision on their explanations. NeSy-XIL is the SOTA neural baseline in the CLEVR-Hans dataset. We used αILP [91] as a differentiable ILP baseline for both tasks, and we used the mode declarations [64], a commonly used language bias in ILP. The perception networks (YOLO and slot attention) are also used for NEUMANN and αILP in Kandinsky patterns and CLEVR-Hans, respectively. More details about each baseline are in App. C.

Result. We show the accuracy in the test split of the Kandinsky patterns in Tab. 3. Both αILP and NEUMANN achieved perfect accuracies for each pattern, although CNN's over-fit while training and performed poorly with testing data. In (P5), the αILP produced *run-out-of-memory*[3] because the pattern involves *seven* objects, and many ground atoms and clauses are produced. NEUMANN can perform scalable visual reasoning beyond tensor-based reasoners. Moreover, we compare the memory consumption of NEUAMANN and αILP on Kandinsky patterns in Tab. 4. NEUMANN clearly consumes less memory than αILP for each pattern, *e.g.* NEUMANN's reasoning graph size

[3] It is not trivial to distribute the tensor-based forward reasoners to several GPUs because each of the instances requires the whole *index tensor*, *i.e.* if we split the large index tensor to distribute, each distributed reasoner cannot refer some atoms and clauses, and thus the reasoning result will be incomplete. Tensor parallelism requires non-trivial engineering [67, 103].

Table 3 Efficiency of NEUMANN does not sacrifice accuracy. The mean classification accuracy in the test split in the Kandinsky patterns dataset over 5 random seeds. OOM denotes out-of-memory on a single GPU. Both αILP and NEUMANN achieved perfect accuracies for each pattern, although CNN's over-fit while training and performed poorly with testing data. As we mentioned in main text, for pattern P5, we evaluated αILP and NEUMANN with a given program and no structure-learning has been performed. Best results are bold.

Model	(P1) pairs	(P2) close	(P3) red tri.	(P4) online	(P5) line
CNN	50.0	52.33	55.0	50.59	-
YOLO+MLP	99.0	72.93	82.95	80.18	-
αILP	**100.0**	**100.0**	**100.0**	**100.0**	OOM
NEUMANN	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**

Table 4 NEUMANN is memory efficient and scales beyond tensors. Memory consumptions on Kandinsky patterns and the visual ILP dataset are shown. The ratio is graph_size/tensor_size. OOM denotes out-of-memory on a single GPU.

Dataset	NEUMANN graph size (ratio)	αILP tensor size
two pairs	**6957** (12.4%)	56064
closeby	**1173** (75.8%)	1548
red-tri.	**4079** (3.12%)	130410
online-p.	**15637** (3.13%)	498960
long-l.	**55632** (NaN %)	OOM

Table 5 Accuracy for CLEVR-Hans dataset compared to baselines over 5 random seeds. For αILP and NEUMANN, we report the mean over the *three* ILP problems. • denotes best, ○ denotes second best result.

Model	Validation	Test
CNN	○99.55	70.34
NeSy	98.55	81.71
NeSy-XIL	•100.00	91.31
αILP	97.5	•**97.52**
NEUMANN	96.67	○**97.43**

is just 3.12% of the tensor size produced by αILP. These results show the memory efficiency of the message-passing reasoners of NEUMANN.

Tab. 5 shows the results for the CLEVR-Hans dataset. As one can see, NEUMANN achieved high accuracy similar to αILP, outperforming neural-based baselines, showing the capability of NEUMANN in complex 3D visual scenes. Moreover, CLEVR-Hans is a confounded dataset, *e.g.* for the first pattern (C1), a large *gray* cube and a large cylinder appear in the training and validation scenes, but in the test scenes, the large cube can be different colors. Thus, neural baselines perform poorly on the test split because the pure data-driven neural models can be easily confounded [97]. Both αILP and NEUMANN achieved high accuracy in the test split because the *downward refinement*-based clause generation can control the generality of clauses, *i.e.* prevent generating too specific clauses to avoid over-fit. This is achieved by trying the small number of search steps in the clause generation and increasing it step by step by checking the performance in the validation split.

We observed peaked weight distributions after training, *i.e.* only one clause gets a large weight. The classification rules obtained by discretizing the clause weights by taking *argmax* for Kandinsky patterns and CLEVR-Hans are shown in Fig. 9. NEUMANN discovered explanatory clauses for each visual pattern. This shows that NEUMANN's learning algorithm can find proper classification rules given positive and negative examples as complex visual scenes.

```
% A pair with the same shape and color, and another
% pair with the same shape and different colors.
kp1(X):-in(O1,X),in(O2,X),in(O3,X),in(O4,X),
        same_shape_pair(O1,O2),same_color_pair(O1,O2),
        same_shape_pair(O3,O4),diff_color_pair(O3,O4).
% Two objects closely located.
kp2(X):-in(O1,X),in(O2,X),closeby(O1,O2).
% A red triangle close to a non-triangle and non-red object.
kp3(X):-in(O1,X),in(O2,X),closeby(O1,O2),
        color(O1,red),shape(O1,triangle),
        diff_shape_pair(O1,O2),diff_color_pair(O1,O2).
% Five objects aligned on a line, and a pair with the same
% shape and color.
kp4(X):-in(O1,X),in(O2,X),in(O3,X),in(O4,X),in(O5,X),
        same_shape_pair(O1,O2),same_color_pair(O1,O2),
        online(O1,O2,O3,O4,O5).

% There is a large cube and a large cylinder.
ch1(X):-in(O1,X),in(O2,X),size(O1,large),shape(O1,cube),
        size(O2,large),shape(O2,cylinder).
% There is a small metal cube and a small sphere.
ch2(X):-in(O1,X),in(O2,X),
        size(O1,small),material(O1,metal),shape(O1,cube),
        size(O2,small),shape(O2,sphere).
% There is a large blue sphere and a small yellow sphere.
ch3(X):-in(O1,X),in(O2,X),size(O1,large),
        color(O1,blue),shape(O1,sphere),
        size(O2,small),color(O2,yellow),shape(O2,sphere).
```

Fig. 9 Clauses discovered by NEUMANN for Kandinsky patterns and CLEVR-Hans. The first 4 clauses for Kandinsky patterns and the last 3 clauses for CLEVR-Hans.

4.3 Visual Reasoning Behind the Scenes

To answer **Q3**, we compare the performance of NEUMANN with conventional differentiable forward reasoners on the behind-the-scenes task showing that (i) it can learn from small data, (ii) it can handle complex visual scenes, (iii) it can learn explanatory programs, and (iv) it can reason about non-observational scenes. Moreover, we also compare the running time of the clause search with a differentiable ILP benchmark.

The task consists mainly of *two* parts: **(Task 1)** learning abstract operations, and **(Task 2)** solving queries with *imagination*, as shown in Fig. 1. We describe each task in detail.

4.3.1 Task 1: Learning Abstract Operations - CLEVR-List

The first task is inductive logic programming from visual scenes for list operations: the *member*, *delete*, and *sort* functions. This is a 3D visual realization of ILP tasks with structured examples, where the goal is to learn abstract list operations given observed input-output pairs, which has been a long-standing

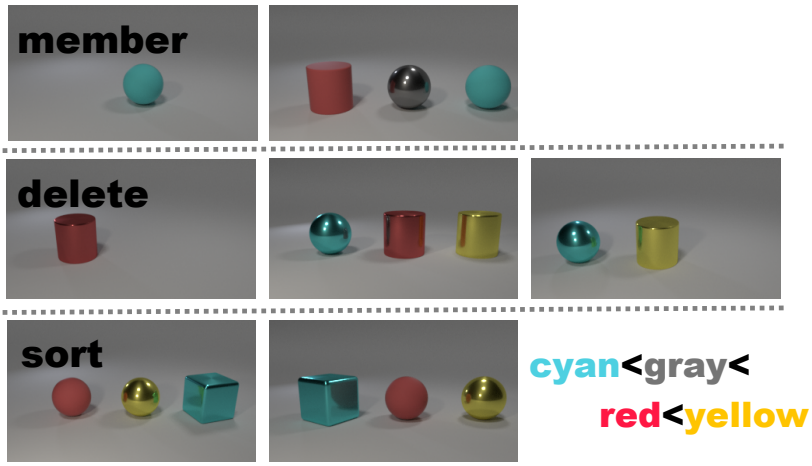


Fig. 10 Task 1: Abstract Program Induction from Visual Scenes (CLEVR-List). Positive examples for abstract list operations of *member*, *delete*, and *sort* (with an order of colors: *cyan* < *gray* < *red* < *yellow*, alphabetical order). Each example consists of several visual scenes representing the input and output of the target programs to be learned. The agents need to handle multiple visual scenes, understanding them deely and comparing each other. (Best viewed in color)

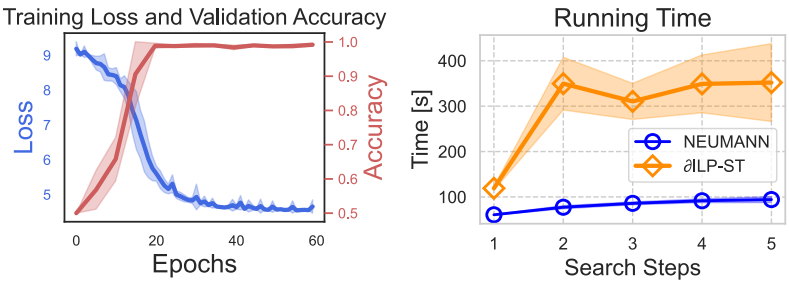


Fig. 11 NEUMANN achieves robust and fast learning. Training loss and validation accuracy for learning the *delete* operation by NEUMANN (left), and comparison of learning time of logic programs against ∂ILP-ST [90], measured for each clause generation step (right). Computed for 5 different random seeds. (Best viewed in color)

task in classical symbolic ILP settings [89,13], and addressed in the differentiable ILP setting recently [90].

We propose *CLEVR-List*, a visual realization of the list-program induction task by using the CLEVR environment [38], which allows users to generate visual scenes that contain multiple objects with different properties, *e.g. large red rubber sphere* and *small gray metal cube*. Fig. 10 shows positive examples for the member, delete and sort functions, which consist of several images representing the inputs and outputs of the target programs to be learned. For example, the first row in the figure represents a positive example for `member`(■, [■, ●, ■]). In each training, validation, and test split, we used 200 examples for each positive and negative label, respectively.

```
% member(X,Y) means X is an element of Y.
member(X,[X|Y]):-.
member(X,[Y|Z]):-member(X,Z).

% delete(X,Y,Z) means Z is the result of deleting X from Y.
delete(X,[X|Y],Y]):-.
delete(X,[Y|Z],[Y|V]):-delete(X,Z,V).

% sort(X,Y) means Y is the result of sorting X.
is_sorted([X,Y|Z]):-smaller(X,Y),is_sorted([Y|Z]).
is_sorted([X]):-.
sort(X,Y):-permutation(X,Y),is_sorted(Y).
```

Fig. 12 Abstract operations discovered by NEUMANN for CLEVR-List. NEUMANN learns abstract list operations from visual scenes dealing with functors.

Compared to the previously addressed visual reasoning benchmarks, *e.g.* Kandinsky patterns and CLEVR-Hans, CLEVR-List is challenging in the sense that the agents need to handle multiple visual scenes as input, *i.e.* deeply understanding them and comparing each other. The list programs are involved with functors, and thus, models need to have the capacity to deal with a large number of ground atoms produced by functors. CLEVR-List requires the following model-building abilities: learning from small data, handling several visual scenes, and learning explanatory programs.

Models. We compare the performance of NEUMANN with ∂ILP-ST [90], which is a state-of-the-art differentiable ILP solver dealing with functors. The parameters for the clause generation (N_{trial}, N_{sample}) are set to $(5, 10)$ for NEUMANN, and we used the same setting for beam search in ∂ILP-ST. We performed 50 epochs of weight optimization using the RMSProp optimizer with a learning rate of $1e-2$ and infer step $T = 5$ for both models. We used mode declarations [64] as a language bias.The number of nested functors is at most 3, discarding lists with duplicated elements. More details including used mode declarations are in App. B.

Result. Fig. 11 (left) shows the training loss and validation accuracy of NEUMANN in the *delete* task, showing the progress of the classification-loss minimization by gradient descent. For five different random seeds, NEUMANN achieved stable learning by producing small training loss and high validation accuracy.

Fig. 11 (right) compares the running time of structure learning of NEUMANN and ∂ILP-ST. We measured the running time of each clause generation step. NEUMANN achieved faster learning using gradient-based scoring and differentiable sampling. As highlighted in Tab. 2, ∂ILP-ST performs exact scoring for each clause. As the search gets deeper, a large number of clauses tend to be generated because of the large number of combinations of symbols. Thus, the running time of ∂ILP-ST increases drastically in the search, but NEUMANN consistently achieved fast structure learning.

We observed peaked weight distributions after training, *i.e.* only some clauses with large weights. Fig. 12 shows the logic programs learned by NEUMANN, obtained by discretizing clause weights using *argmax* after training. NEUMANN produced explanatory programs that achieved 1.0 of accuracy in the test split for each operation. These results show that NEUMANN solved the proposed **(T1)** CLEVR-List task, outperforming a differentiable ILP baseline by the running time.

4.3.2 Task 2: Reasoning on Behind-the-Scenes

The second task is to perform visual reasoning given queries where the answers are derived by the *reasoning behind the scenes*, *e.g.* the agent needs to think of the non-observational scenes with *imagination*. We consider 4 abstract operations: *delete*, *append*, *reverse*, and *sort*. The input is a pair of an image and a query represented as an atom. For example, a query *"What is the color of the second left-most object after deleting a gray object?"* can be represented as a query atom: `query(q_delete, gray, 2nd)`, where `q_delete` is a constant that represents the query type about the deletion. Forward reasoners can solve this task by combining clauses to parse input visual scenes and derive answers. We used 40k questions (10k for each operation) associated with 10k visual scenes. We compare the performance of NEUMANN and αILP [91].

Image Generation. We generated visual scenes using the CLEVR environment [38]. Each visual scene contains at most 3 objects with different attributes: (i) colors of *cyan, gray, red,* and *yellow*, (ii) shapes of *sphere, cube,* and *cylinder*, (iii) materials of *metal* and *matte*. We excluded color duplications in a single image.

Query Generation. We generated queries for the dataset using query templates, which produce various queries using different features of objects. We used the following template: *"What is the color of the [Position] object after [Operation]?"*, where [Position] can take either of: *left-most first*, *second*, or *third*. [Operation] can take the following form: (i) *delete* an object, (ii) *append* an object to the left, (iii) *reverse* the objects, and (iv) *sort* the objects with an order of colors: *cyan* $<$ *gray* $<$ *red* $<$ *yellow* (alphabetical order). Fig. 13 shows examples of an input scene and some paired queries with their answers. More examples of input scenes, queries and their answers are in App. E.

Models. We used the clauses in Fig. 14 for NEUMANN and αILP. The first clause about `chain` generates a chain of colors given a visual scene. For example, given the visual scene in Fig. 13, the atom `chain`([, , ]) is deduced using body atoms `left_of`( , ) and `left_of`(, ). The second clause parses the chained objects to colors. For example, the atom `scene`([`red`, `gray`, `cyan`]) is deduced using body atoms of `chain`([, ,]), `color`(, `red`), `color`(, `gray`), and `color`(, `cyan`). The last 4 clauses compute answers for different types of queries using the parsed `scene` atoms, list operations, and other utility predicates (*cf.* Fig. 18 in the appendix). `query2` and `query3` represent queries, *e.g.* `query2(q, q_sort, 2nd)` represents a query *"What is the*

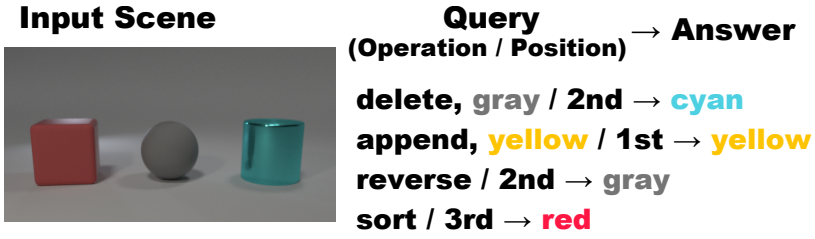


Fig. 13 Task 2: Visual Reasoning Behind the Scenes. The task is to compute the answers for given queries paired with input visual scenes. A query consists of an *operation* and a target *position*, and an answer is a color, *e.g.* a query *"What is the color of the 2nd left-most object after deleting a gray object?"* whose answer is *cyan*. (Best viewed in color)

```
% generate a chain of objects from an image
chain([Object1,Object2,Object3]):-
              left_of(Object1,Object2),left_of(Object2,Object3).

% parse the visual scene as a list of colors
scene([Color1,Color2,Color3]):-chain([Object1,Object2,Object3]),
              color(Object1,Color1),
              color(Object2,Color2),
              color(Object3,Color3).

% answer the delete query
answer(X):-scene(Colors1),delete(Color,Colors1,Colors2),
    query3(q_delete,Color,Position),get_color(Colors2,Position,X).

% answer the append query
answer(X):-scene(Colors1),append([Color],Colors1,Colors2),
    query3(q_append,Color,Position),get_color(Colors2,Position,X).

% answer the reverse query
answer(X):-scene(Colors1),reverse(Colors1,Colors,Colors2),
    query2(q_reverse,Position),get_color(Colors2,Position,X).

% answer the sort query
answer(X):-scene(Colors1),sort(Colors1,Colors2),
    query2(q_sort,Position),get_color(Colors2,Position,X).
```

Fig. 14 Clauses to answer queries for behind-the-scenes. The first clause about `chain` generates a chain of objects given a visual scene. The second clause parses the chained objects to colors. The last 4 clauses compute answers for different types of queries using the parsed `scene` atoms, list operations, and other utility predicates.

color of the 2nd-left object after sorting the objects?". A query atom is given being paired with an input image.

Result. Tab. 6 shows the accuracy for each type of queries. αILP produced *run-out-of-memory*, however, NEUMANN successfully solves different types of queries with high accuracy. As shown in Fig. 14, the clauses to answer the queries consist of many functors for the list representation and existentially quantified variables, and thus a large number of ground atoms and clauses is

Table 6 NEUMANN can reason behind the scenes beyond tensor-based reasoners. Classification accuracy for behind-the-scenes tasks. OOM denotes out-of-memory on a single GPU.

	delete	append	reverse	sort
αILP	OOM	OOM	OOM	OOM
NEUM.	**0.98**	**0.99**	**0.98**	**0.98**

Table 7 Many ground representations are required for reasoning about behind-the-scenes. Number of ground atoms $|\mathcal{G}|$ and ground clauses $|\mathcal{C}^*|$ for each task. For Kandinsky patterns, the mean value for different patterns (P1)-(P4) is shown.

	Kandinsky	CLEVR H.	Behind-the-Scenes
#Atoms	131	165	150K
#Clauses	288	90	1.1M

generated, which is difficult to be handled by the conventional tensor-based reasoner, *i.e.* ∂ILP, ∂ILP-ST, and αILP. In fact, Tab. 7 shows the numbers of ground atoms and clauses generated for Kandinsky patterns, CLEVR-Hans, and Behind-the-Scenes, respectively. Behind-the-Scenes requires the models to handle many ground representations, *i.e.* a large *Herbrand base* with functors, and thus memory-efficient reasoning is necessary. These results show that NEUMANN solved the Behind-the-Scenes task outperforming conventional tensor-based reasoners overcoming the bottleneck of the intensive memory consumption scaling to deal with functors on visual scenes and query answering.

4.4 Advantages against other Symbolic and Neuro-Symbolic Methods

To answer **Q4**, we compare the performance of NEUMANN against state-of-the-art symbolic and neuro-symbolic methods. Moreover, we show that NEUMANN can produce visual explanations efficiently using gradients using the end-to-end differentiable reasoning architecture.

4.4.1 Scalable Visual Reasoning and Learning

First, we show that NEUMANN can perform scalable visual reasoning, *i.e.* it can handle a large number of examples of complex visual scenes. To show that, we compare the inference time for Kandinsky patterns and CLEVR-Hans. We used *two* state-of-the-art neuro-symbolic methods to be compared:

– **Feed-Forward Neural-Symbolic Learner (FFNSL) [17]** is a neuro-symbolic learning framework that integrates Answer Set Programming (ASP) with neural networks. It performs visual perception using neural networks, reads out their output as weighted logic representations, and performs Inductive Learning of Answer Set Programs (ILASP) [49], which conducts efficient structure-learning of logic programs based on ASP semantics. It uses CLINGO [31], a well-established ASP-solving system for inference. FFNSL can handle noisy input as weighted expressions, but its inference engine requires discrete input and returns discrete logic representations.

– **DeepProbLog [58]** is a neuro-symbolic framework that integrates neural networks to ProbLog [24], which is a well-established probabilistic logic inference engine. ProbLog accepts probabilistic input and produces probabilistic output by performing exact probabilistic inference by compiling logic representations into circuits, *e.g.* Sentential Decision Diagrams [21]. DeepProbLog obtains gradients out of the ProbLog output and train neural networks efficiently, and it is also applied to perform structure learning in the *sketching* setting [95,8], where programs are learned to complete partially-given input programs.

We measured the inference time (including the grounding of programs) for the training split. We changed the proportion of the data to be used from 0.2 (use 20% of the training data) to 1.0 (use the full training data) for each dataset. We used a batch size of 200 consistently throughout the experiments. We set 1000 seconds as the timeout.

Fig. 15 shows the inference-time comparison of NEUMANN, FFNSL, and DeepProbLog on Kandinsky patterns and CLEVR-Hans. In each dataset, NEUMANN achieved the fastest inference among the baselines. Especially in the patterns that require complex classification rules and many ground atoms, *e.g.* red-triangle and online-pair, NEUMANN significantly outperformed DeepProbLog and FFNSL. On online-pair, DeepProbLog timed out even with 20% of training data. This shows that NEUMANN can perform scalable visual reasoning for a large amount of data. We note that NEUMANN and DeepProbLog achieve differentiable reasoning, *i.e.* we can obtain gradients out of the probabilistic reasoning result, but FFNSL does not provide this function because of its pure-symbolic reasoner.

Moreover, to show that NEUMANN performs scalable visual learning, we compare the performance and running time of learning of NEUMANN and FFNSL using Kandinsky and CLEVR-Hans. In this setting, FFNSL serves as a *symbolic* learning benchmark because it uses symbolic ILP systems, *e.g.* ILASP [49], to learn programs.

We changed the proportion of the training and validation data: 0.01 (1 %), 0.05 (5 %), 0.1 (10 %), and 0.5 (50 %). We measured the test accuracy using the full test data (100 %) consistently for each setting. We measured the running time of learning, which includes the whole process, *i.e.* visual perception, obtaining logic representations, and search logic programs. We used a batch size of 200 and a learning rate of $1e-2$, and trained 20 epochs. We set (N_{trial}, N_{sample}) as $(4, 10)$ for red-triangle in Kandinsky and $(6, 10)$ for the third pattern of CLEVR-Hans. To achieve FFNSL on Kandinsky and CLEVR-Hans, we convert each visual scene to a weighted example for ILASP as described in [17]. We set to time out as 5000 seconds.

Fig. 16 shows the accuracy of the test split (left) and the learning time (right). In both datasets, FFNSL achieved high accuracy with a very small number of training data (1% of training visual scenes), showing the advantage of the symbolic learning method to generalize from small data. However, as the number of training data increases, the learning time increases drastically.

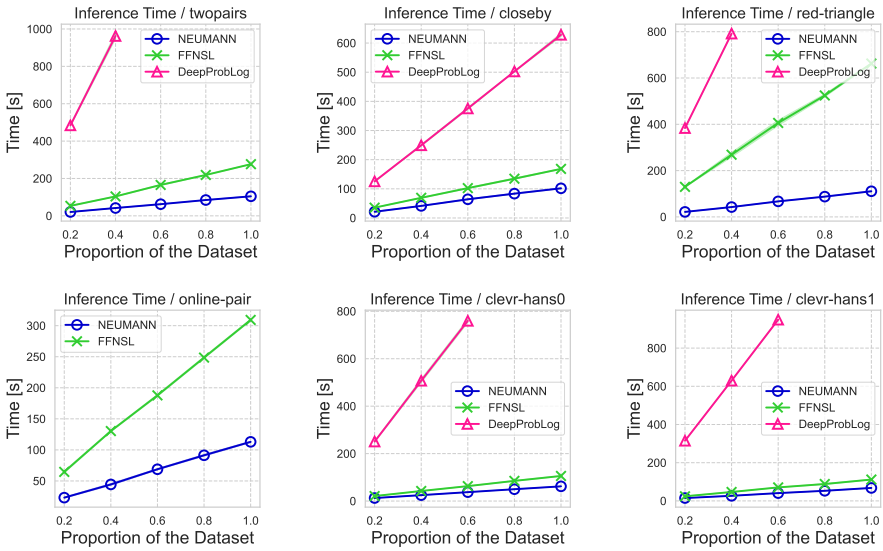


Fig. 15 NEUMANN performs scalable visual reasoning. We compare the inference time of NEUMANN, FFNSL [17] (using CLINGO [31]), and DeepProbLog [58] (using ProbLog [24]) on Kandinsky and CLEVR-Hans using the training split (20k images for each Kandinsky pattern, and 9k images for each CLEVR-Hans class). NEUMANN and DeepProbLog compute probabilistic output, and FFNSL computes discrete output. As we mentioned in main text, the time-out was set to 1000 seconds. In online-pair, DeepProbLog timed out even with 20% of training data. Computed for 5 different random seeds. (Best viewed in color)

To this end, FFNSL handled less than 10% of the training visual scenes in Kandinsky and CLEVR-Hans. In contrast, NEUMANN performed learning much faster than FFNSL using a large number of training visual scenes. This shows that NEUMANN is scalable for large datasets, *i.e.* it can learn from a large number of visual scenes. Moreover, in Kandinsky, although NEUMANN produced test accuracy with a large variance with a small number of training data, it achieved higher accuracy and gained stability with less variance by using more data. Overall, NEUMANN outperformed FFNSL in terms of learning time, allowing the system to use more training data and producing competitive test accuracy.

4.4.2 Explanations by Gradients

We show that NEUMANN can produce gradient-based explanations efficiently by working with neural networks seamlessly. We use *input gradients* [4], which is a widely-used explanation method for any differentiable models. Input gradients are gradients with respect to input, *i.e.* for a differentiable function f and an input $\mathbf{x}$ and output y, it computes $\mathbf{e} = \partial y / \partial \mathbf{x}$, where each element $\mathbf{e}$

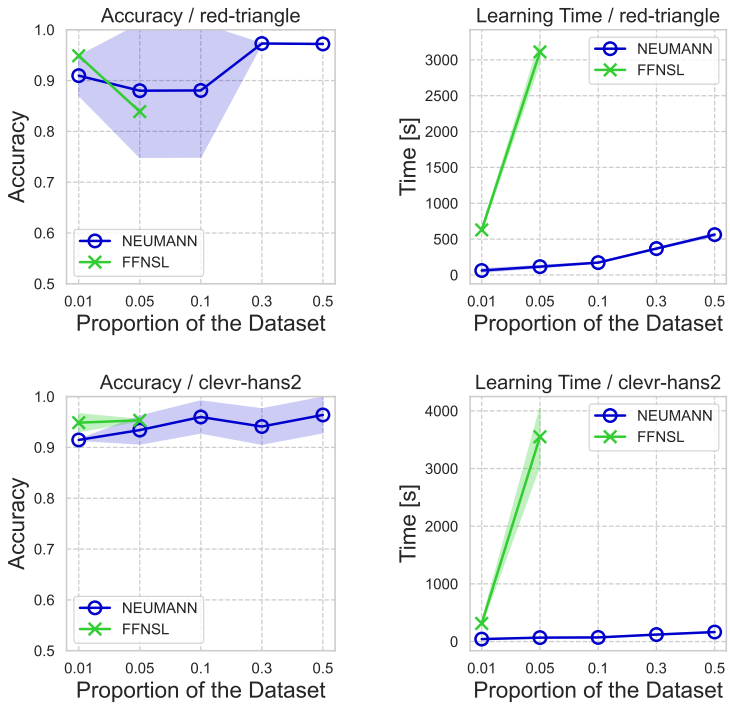


Fig. 16 NEUMANN performs scalable learning on complex visual scenes. We compare the performance and learning time of NEUMANN and FFNSL [17] (using ILASP [49], which serves as a symbolic-learning benchmark), on Kandinsky patterns and CLEVR-Hans by changing the proportion of training/validation data. The accuracy is measured on the full test data. As mentioned in main text, the time-out was set to 5000 seconds and FFNSL timed out with more than 10% of training data. Computed for 3 different random seeds. (Best viewed in color)

represents how the corresponding input (*e.g.* a pixel) is effective to the output[4]. NEUMANN can produce input gradients over input atoms, *i.e.* we compute:

$$\mathbf{e}_{atoms} = \frac{\partial y}{\partial \mathbf{x}^{(0)}_{atoms}} = \frac{\partial \mathbf{x}^{(T)}_{atoms}}{\partial \mathbf{x}^{(0)}_{atoms}} \cdot \frac{\partial y}{\partial \mathbf{x}^{(T)}_{atoms}}. \tag{11}$$

Since the reasoning function of NEUMANN is end-to-end differentiable, $\mathbf{e}_{atoms}$ can be computed efficiently using automatic differentiation (AD), *i.e.* just calling the backward function once after the reasoning[5].

[4] For simplicity, we do not take the element-wise product $\mathbf{x} \odot \frac{\partial y}{\partial \mathbf{x}}$.

[5] Since $\mathbf{x}^{(0)}_{atom}$ is not a leaf node in the computational graph, we prepare a dummy variable $\mathbf{z}_0$ with the same shape as $\mathbf{x}^{(0)}_{atoms}$ and all elements are initialized with 0. In the forwarding, we compute $\mathbf{x}^{(0)}_{atoms} = \mathbf{x}^{(0)}_{atoms} + \mathbf{z}_0$, and extract gradients stored in $\mathbf{z}_0$ after calling the backward function.

Let $\mathbf{M}_1, \ldots, \mathbf{M}_n$ be (attention) masks over n objects for an input scene produced by an object-centric perception network. We compose visual explanations as follows:

$$E(\mathbf{x}) = \sum_i \phi_i \cdot \mathbf{M}_i \tag{12}$$

where ϕ_i is a weight for the i-th mask, which is computed as the maximum probability of input ground atoms regarding the i-th object. Let $\mathcal{J} = \{j_1, \ldots, j_m\}$ be indices of the ground atoms regarding the i-th object (*e.g.* $\texttt{color}(\texttt{obj}_\texttt{i}, \texttt{red})$) in ordered set of ground atoms $\mathcal{G}$. We compute the weight ϕ_i for the mask $\mathbf{M}_i$:

$$\phi_i = \max(\mathbf{e}_{atoms}[j_1], \ldots, \mathbf{e}_{atoms}[j_m]), \tag{13}$$

where $\mathbf{e}_{atoms}$ is computed by Eq. 11. For example, if `obj1` is a key factor for the classification, the corresponding attention mask $\mathbf{M}_1$ gets a large weight, *i.e.* highlights `obj1`.To this end, Eq. 12 computes a heatmap that highlights only objects which are effective in the reasoning result.

Fig. 17 shows explanations produced by NEUMANN for CLEVR-Hans using clauses listed in Fig. 9. For each pair of images, the left one shows the original input and the right one shows the heatmap. NEUMANN successfully produced explanations highlighting objects which are the factors for the classification. For example, the first class is about a *large cube* and a *large cylinder*, and NEUMANN highlights both but not others for each input. The explanation can be completed very efficiently by using automatic differentiation (AD).

The same explanation could be obtained by using gradients in DeepProbLog. However, as shown in Section 4.4.1, it has a scalability issue for a large amount of complex visual scenes. In contrast, NEUMANN produces gradient-based explanations but still can handle a large amount of complex visual scenes in a scalable manner. Moreover, FFNSL relies on the discrete inference engine, CLINGO, and thus it is difficult to produce gradient-based explanations using automatic differentiation, *i.e.* it requires additional hard coding for explanations. Overall, NEUMANN is the only framework that achieves scalable differentiable forward reasoning and learning, producing explanations on complex visual scenes working with neural networks efficiently.

4.5 Discussions

We now discuss NEUMANN's advantages, computation, impact, and limitations.

What are the advantages compared to pure-symbolic learners?

The most promising feature of NEUMANN compared to pure symbolic systems is its capability to handle a large amount of visual input in a scalable manner. As shown in Sec. 4.4, NEUMANN can perform visual reasoning and learning, outperforming state-of-the-art neuro-symbolic benchmarks regarding

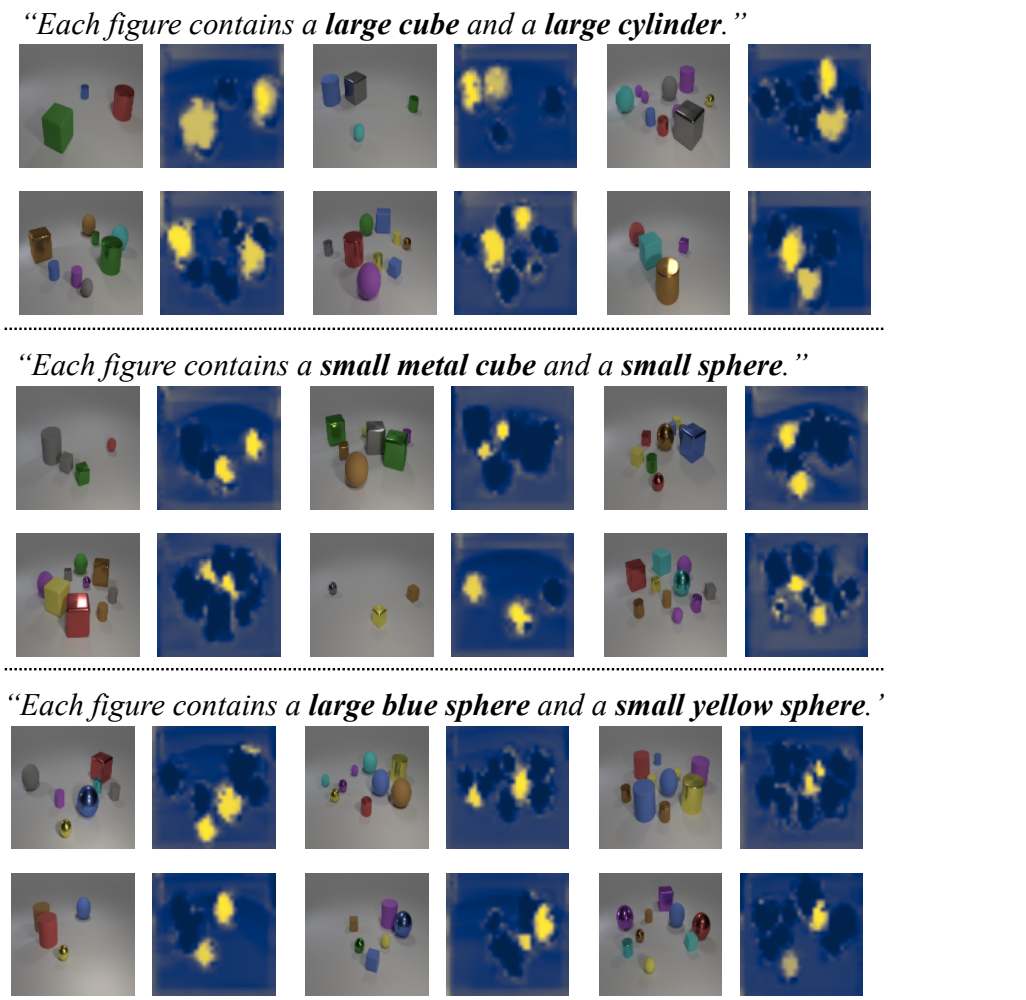


Fig. 17 NEUMANN can explain its reasoning using gradients. Explanations produced by NEUMANN by computing input gradients [4] for input probabilistic atoms $\mathbf{x}^{(0)}_{atoms}$ and visualizing together with relevant attention maps. (Best viewed in color)

running time and performance. This feature is crucial for tightly integrating learning and reasoning with neural networks, *e.g. algorithmic supervision* [74] where neural networks are trained efficiently using gradients via symbolic algorithms. For such a setting, reasoners should be able to conduct scalable reasoning for a large amount of data to train neural networks. Otherwise, they would be the bottleneck, limiting the applicability of the neuro-symbolic systems.

Moreover, as shown in Sec. 4.4.2, NEUMANN can use gradient-based XAI methods to produce visual explanations working with perception networks efficiently, and it is difficult to produce the same result with a pure symbolic

system without additional hard coding. This feature of NEUMANN leads to essential applications, *e.g.* the *right for the right reasons* [83] approach, which trains neural networks to produce correct explanations using the gradient-based explanations.

Overall, NEUMANN does not contradict symbolic approaches but provides a basis for better neuro-symbolic systems. It has been reported that a pure symbolic system can handle noisy examples [50], refuting the motivation of ∂ILP [29]. However, NEUMANN provides many other benefits of scalable reasoning and learning paradigm that is compatible with gradient-based methods that include a significant part of the success of DNNs.

What makes NEUMANN's reasoning and learning scalable?

The scalable performance of NEUMANN can be explained by two reasons. (1) NEUMANN *grounds programs once*, then use the resulting computational graph repeatedly, as other differentiable forward reasoners do [29,90,91]. It means that NEUMANN does not compute logic operations (*e.g.* unification) for each specific query. Instead, NEUMANN performs forwarding on the computational graph and then obtains the results. In contrast, (differentiable) backward reasoning, employed in DeepProbLog [58], needs to construct a new computational graph for a new query, making the reasoning expensive. (2) More importantly, *NEUMANN is a graph neural network and performs reasoning on GPUs*[6]. The ground-once scheme enables the reasoner to build and fix the computational graph as users do with neural networks, *i.e.* defining the network architecture and a set of weights, then the computational graph is constructed and fixed so forwarding can be conducted on GPUs. When dealing with a batch of examples (*e.g.* 200 examples), NEUMANN can process them in parallel very efficiently. This feature is not trivial for logic reasoners. Typically, they process a batch of examples sequentially. For instance, DeepProbLog uses Sentential Decision Diagrams [21] for its reasoning, and it requires building different SDDs on CPUs for each query, and FFNSL uses the CPU-based reasoner (CLINGO [31]). Thus it requires non-trivial efforts for these reasoners to compute reasoning by using GPUs in a scalable manner.

Why is it crucial to improve the memory consumption of differentiable forward reasoners?

Differentiable forward reasoning, introduced in the ∂ILP framework [29], encodes logic programs to *tensors*. Differentiable forward reasoning is inherently memory intensive and thus limits the expressivity of logic programs, *e.g.* no functors are allowed, each rule consists of at most 2 body atoms, and each predicate takes at most 2 arguments. ∂ILP has been extended to handle structured programs with functors [90], by incorporating search techniques in ILP [64,71,16], leading to αILP [91], which learns logic rules from complex visual scenes. However, these methods inherit the memory-intensive tensors and thus cannot handle complex programs for abstract visual reasoning. Differentiable forward reasoning gains several advantages compared to pure symbolic reasoners, as discussed in the previous paragraph. Thus it is crucial to improve

[6] NEUMANN is implemented using PyG, an established GNN library `https://pyg.org/`

memory consumption so that a wide range of programs can be handled in the framework to expand the applicability of neuro-symbolic approaches.

Why is graph encoding more efficient than dense tensor encoding?

Graphs can represent the relations between different atoms more efficiently than dense tensors. For example, suppose we want to encode the following information, *"Atom X is not deduced by atom Y"*. A reasoning graph can represent this information simply by not having edges between X and Y. In contrast, the dense tensors in differentiable ILP frameworks [29,90,91] need to hold *false symbol* $\bot$ for each combination of X and Y, *i.e.* the system needs to keep all of the combinations of ground atoms. This results in NEUMANN being memory efficient as shown in Proposition 1.

Can sparse tensors result in reasoners comparable to message-passing reasoners?

Sparse tensors cannot account for differentiable forward reasoning because they do not support some essential tensor operations as they compress the tensors by breaking the row-and-column structure. As shown in ∂ILP [29], differentiable forward reasoning uses the *slice* and *gather* operations on tensors repeatedly, and those are not supported by sparse tensors. Thus simply using sparse tensors does not lead to memory-efficient differentiable forward reasoners.

Why can existing VQA models not solve the Behind-the-Scenes task?

VQA models accept input as a tuple of an image and a question in natural-language sentences [2]. Symbolic programs, typically described as a Domain Specific Language (DSL) to compute answers, are generated by parsing the input question using neural networks. Given visual scenes, it is unclear how to perform program induction on their DSL since there is no uniform structure-learning algorithm for each DSL. Thus simply using existing VQA models for the proposed task cannot be a solution.

Limitations. Although NEUMANN is a more general framework compared to classic symbolic and neuro-symbolic frameworks, it does suffer from some limitations: (1) The language to be handled is limited to definite clauses, which are rules in FOL with a single head atom. Symbolic systems can handle more complex structures, *e.g.* choice rules in Answer Set Programming (ASP) systems [10]. (2) The learning algorithm is not jointly training perception networks and logic programs. (3) The message-passing algorithm is not connected to well-known probabilistic semantics, *e.g.* distribution semantics [86].

5 Related Work

NEUMANN builds upon different sub-fields of AI. We revisit relevant studies.

Symbolic AI. Symbolic representations, *e.g.* First-Order Logic (FOL), provide essential functions of knowledge representation and reasoning capabilities to AI systems, which are difficult to be provided by purely neural-based models [5,9,70]. A pioneering study of inductive inference was done

in the early 70s [75]. Many systems have been developed for *inductive inference* [1], *e.g.* Model Inference System (MIS) [89] has been implemented as an efficient search algorithm for logic programs. Inductive Logic Programming (ILP) [64,71,16] has emerged at the intersection of machine learning and logic programming. ILP systems using Answer Set Programming (ASP) can learn logic programs beyond definite clauses [49,48], *e.g.* choice rules. ILP has advantages compared to data-driven DNNs, *e.g.* it can learn explicit programs and learn from small data. Thus, combining ILP with DNNs is a promising approach to overcoming the limitations of the current data-driven machine-learning paradigm. NEUMANN embraces the symbolic learning approaches in the neuro-symbolic setting, where logical reasoning and neural learning are tightly integrated.

Probabilistic Logic and Neuro-Symbolic AI. Combining probabilities with symbolic logic has been addressed to establish reasoning systems that can handle uncertainty, *e.g.* distribution semantics [86] and Bayesian logic programs [41]. Probabilistic Inductive Logic Programming [23] combines ILP with probabilistic semantics establishing a new learning paradigm. Structure learning algorithms for probabilistic logic programs have been developed, *e.g.* SLIPCOVER [7]. These approaches focus on learning with probabilistic semantics, but NEUMANN engages differentiable reasoning and learning, where parameters get gradients optimized via gradient descent. Lifted inference [100] addresses efficient reasoning, *e.g.* reducing computational graphs by using symmetry, and these techniques could be incorporated into NEUMANN since it employs graphs as its representation. Markov Logic Networks (MLNs) [80] takes a similar approach to ground the logic programs to produce a graph structure. MLNs perform inference based on Bayesian networks, but NEUMANN computes differentiable forward reasoning by message-passing as graph neural networks.

Integration of symbolic computations and neural networks, called *neuro-symbolic AI* [30,40], has attracted a lot of attention in recent years. Many frameworks have been developed for parameter estimation of DNNs using symbolic programs, *e.g.* DeepProbLog [58,59], NeurASP [105], SLASH [94], NS-CL [60], differentiable theorem provers [82], and Embed2Sym [3]. In a similar vein, differentiable structure learners have been developed [29,62,90,91], and NEUMANN extends their capacity by having memory-efficient reasoning and computationally-efficient learning. TensorLog [15] performs message-passing for backward reasoning, but NEUMANN realizes forward reasoning. GNNs have been used for reasoning [78,73] by composing logical expressions as graphs, where neural representations are trained given symbolic knowledge. In contrast, NEUMANN performs structure learning using reasoning graphs and fuzzy logic operations. Logical Neural Networks (LNNs) [81] is a class of neural networks where each node has its logical semantics. LNNs parameterize soft-logical operations, but NEUMANN parameterizes clauses with their weights. Lifted Relational Neural Networks [102] uses rules as a template to produce deep neural networks. NEUMANN uses rules as a template to produce differentiable message-passing forward reasoner. Integration DNNs with abductive

learning [20] has been addressed, where the agent learns to complete a symbolic knowledge base. This approach does not address program induction from raw inputs. In contrast, NEUMANN performs structure learning from complex visual scenes. MetaABD [19] has been proposed to perform program induction based on abductive learning by integrating a learning system Metagol [65]. Metagol handles definite clauses without functors, but NEUMANN can learn programs with functors.

Visual Reasoning Datasets. The deep-learning community has developed many image datasets for evaluating different image-recognition models, *e.g.* MNIST [26] and ImageNet [25]. However, these datasets are dedicated to simple label classification, and thus difficult to assess the reasoning abilities of machine-learning models. To overcome this limitation, visual datasets with reasoning requirements have been developed. Visual Question Answering (VQA) [2,104,45] is a well-established scheme to learn to answer questions given as natural language sentences together with input images. VQA has an assumption that the programs to compute answers are given as input questions. However, in Behind-the-Scenes, the agents need to learn abstract programs to compute the answers. Moreover, VQA models do not address learning from small data and transferring the obtained knowledge to new tasks, which are parts of the Behind-the-Scenes requirement. Neuro-symbolic models achieve multi-modal learning on VQA [107,98], but NEUMANN addresses rather structure-learning problems with differentiable logic programming. VQAR [36] is a variant of VQA with relational reasoning, CLEVRER [106] is an extension of CLEVR with sequential input, and MNIST-Addition [58] is about learning DNNs to add hand-written digits. These datasets involve essential aspects of reasoning, *e.g.* , relations with multiple entities, temporal reasoning, and arithmetic computation. However, as shown in Tab. 1, Behind-the-Scenes achieve the *four* essential model-building aspects: (i) learning from small data, (ii) learning from complex visual scenes, (iii) learning explanatory programs, and (iv) reasoning beyond observations. Previously proposed datasets cover some of these aspects but not all of them. The proposed Behind-the-Scenes task is the first dataset to assess the four model-building abilities of machine-learning models. Abstract Visual Reasoning (AVR) has been addressed to test the ability to apply previously gained knowledge and programs in a completely new setting, posing challenges to DNNs [35,57, 14]. The methods have been evaluated on simple tasks of abstract puzzles, *e.g.* Raven's progressive matrices [77]. The proposed task, Behind-the-Scenes, requires structured program induction and reasoning beyond observation in complex 3D visual scenes, which have not been addressed in previous AVR studies.

A motivation of the proposed behind-the-scene task is *problem solving*, which is an essential aspect of human intelligence of solving problems beyond perception using reasoning [69,68]. Humans can learn much from a small number of experiences developing capacities to represent physical objects and reason about their motion [96,6]. Inspired by these studies, the development of adaptive learning skills of humans has been addressed as *model building*

problems [47], and data-driven DNNs are insufficient to achieve these aspects. NEUMANN tackles this challenge by performing memory-efficient differentiable forward reasoning using DNNs.

Graphs and Circuits for Reasoning. Many approaches have been developed to encode the reasoning structures to graphs and circuits. Binary Decision Diagrams (BDDs) [12] encode propositional logic expressions compactly as a directed graph, leading to variant structures, *e.g.* Sentential Decision Diagrams (SDDs) [21,44], Zero-suppressed Decision Diagrams (ZDDs) [61], and Zero-supressed Sentential Decision Diagrams (ZSDDs) [72]. These architectures are developed for propositional logic or combinatorial optimization, but the reasoning graph of NEUMANN represents first-order logic and addresses differentiable reasoning. Their efficient compression algorithms and operations between graphs (*e.g.* taking conjunction between two graphs) for these structures could be applied to reasoning graphs in NEUMANN. Sum-Product Networks (SPNs) [76] encode tractable probability distributions in graphs, which repeatedly consist of sum and product layers. NEUMANN shares a similar structure since its reasoning graph consists of atom nodes to compute disjunctions and conjunction nodes, and performs bi-directional message-passing. SPNs solve exact probabilistic inference, but NEUMANN addresses differentiable reasoning on first-order logic.

6 Conclusion

We presented NEUMANN, a memory-efficient differentiable forward reasoner that passes messages on reasoning graphs. NEUMANN compiles logic programs in first-order logic to a graph that encloses the process of forward reasoning and performs message-passing in a neural fashion. Moreover, we proposed a computationally-efficient learning algorithm combining gradient-based scoring and differentiable sampling of clauses. In our experiments, we have shown: (1) The message-passing reasoning algorithm simulates the differentiable forward reasoning dealing with uncertainty. (2) NEUMANN can solve visual ILP problems combined with DNNs, outperforming neural baselines and consuming less memory than the other differentiable ILP benchmarks. (3) NEUMANN solves the Behind-the-Scenes task outperforming conventional differentiable reasoners, providing model-building abilities beyond simple perception capabilities, *i.e.* learning from small data, understanding visual scenes deeply, learning explanatory programs, and reasoning about non-observational scenes. (4) NEUMANN performs scalable visual reasoning and learning, outperforming state-of-the-art symbolic and neuro-symbolic methods regarding running time and performance. Moreover, NEUMANN can incorporate XAI methods efficiently, *i.e.* NEUMANN produces gradient-based visual explanations using DNNs.

NEUMANN provides several interesting avenues for future work. NEUMANN is an instance of GNNs, providing the capability of representation learning to make neuro-symbolic reasoning more robust and multi-modal. Moreover, NEUMANN enables differentiable reasoning on complex logic pro-

grams with functors and thus can be used for vital applications, such as planning, meta-interpreters, and knowledge-enhanced foundation models. NEUMANN is also promising for the *right for the right reasons* approach [83], where neural networks are trained to produce correct explanations and thus a vital factor to achieve explainable machine learning systems. Generally, it bridges the current data-driven machine learning paradigm to perform problem-solving beyond perception with knowledge representation and reasoning.

References

1. D. Angluin and C. H. Smith. Inductive inference: Theory and methods. *ACM Comput. Surv.*, 15(3):237–269, Sept. 1983.
2. S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh. Vqa: Visual question answering. In *International Conference on Computer Vision (ICCV)*, 2015.
3. Y. Aspis, K. Broda, J. Lobo, and A. Russo. Embed2Sym - Scalable Neuro-Symbolic Reasoning via Clustered Embeddings. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2022.
4. D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K. Müller. How to explain individual classification decisions. *J. Mach. Learn. Res.*, 11:1803–1831, 2010.
5. C. Baral. *Knowledge representation, reasoning and declarative problem solving*. Cambridge university press, Cambridge, 2003.
6. P. W. Battaglia, J. B. Hamrick, and J. B. Tenenbaum. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110(45):18327–18332, 2013.
7. E. Bellodi and F. Riguzzi. Structure learning of probabilistic logic programs by searching the clause space. *Theory Pract. Log. Program.*, 15(2):169–212, 2015.
8. M. Bošnjak, T. Rocktäschel, J. Naradowsky, and S. Riedel. Programming with a differentiable forth interpreter. In *International Conference on Machine Learning (ICML)*, volume 70, pages 547–556, 2017.
9. R. Brachman and H. Levesque. *Knowledge representation and reasoning*. Elsevier, Amsterdam, 2004.
10. G. Brewka, T. Eiter, and M. Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54(12):92–103, 2011.
11. T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. In *Advances in neural information processing systems (NeurIPS)*, pages 1877–1901, 2020.
12. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.
13. R. Caferra. *Logic for Computer Science and Artificial Intelligence*. Wiley-IEEE Press, New York, 2013.
14. G. Camposampiero, L. Houmard, B. Estermann, J. Mathys, and R. Wattenhofer. Abstract visual reasoning enabled by language. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 2642–2646, June 2023.
15. W. W. Cohen, F. Yang, and K. Mazaitis. TensorLog: A probabilistic database implemented using deep-learning infrastructure. *J. Artif. Intell. Res.*, 67:285–325, 2020.
16. A. Cropper, S. Dumančić, and S. H. Muggleton. Turning 30: New ideas in inductive logic programming. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4833–4839, 2020.
17. D. Cunnington, M. Law, J. Lobo, and A. Russo. FFNSL: feed-forward neural-symbolic learner. *Mach. Learn.*, 112(2):515–569, 2023.
18. M. Cuturi and M. Blondel. Soft-dtw: A differentiable loss function for time-series. In *International Conference on Machine Learning (ICML)*, volume 70, pages 894–903, 2017.

19. W. Dai and S. H. Muggleton. Abductive knowledge induction from raw data. In Z. Zhou, editor, *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.
20. W. Dai, Q. Xu, Y. Yu, and Z. Zhou. Bridging machine learning and logical reasoning by abductive learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2811–2822, 2019.
21. A. Darwiche. SDD: A new canonical representation of propositional knowledge bases. In T. Walsh, editor, *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 819–826, 2011.
22. R. Davis, H. Shrobe, and P. Szolovits. What is a knowledge representation? *AI Magazine*, 14(1):17, 1993.
23. L. De Raedt and K. Kersting. *Probabilistic Inductive Logic Programming*, pages 1–27. Springer, Berling, Heidelberg, 2008.
24. L. De Raedt and A. Kimmig. Probabilistic (logic) programming concepts. *Mach. Learn.*, 100(1):5–47, 2015.
25. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.
26. L. Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.
27. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 4171–4186, 2019.
28. V. P. Dwivedi and X. Bresson. A generalization of transformer networks to graphs. *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*, 2021.
29. R. Evans and E. Grefenstette. Learning explanatory rules from noisy data. *J. Artif. Intell. Res.*, 61:1–64, 2018.
30. A. Garcez and L. Lamb. Neurosymbolic ai: the 3rd wave. *Artificial Intelligence Review*, pages 1–20, 03 2023.
31. M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub. Multi-shot ASP solving with clingo. *Theory Pract. Log. Program.*, 19(1):27–82, 2019.
32. S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 4(1):1–58, 1992.
33. W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1024–1034, 2017.
34. K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
35. S. Hu, Y. Ma, X. Liu, Y. Wei, and S. Bai. Stratified rule-aware network for abstract visual reasoning. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 1567–1574, 2021.
36. J. Huang, Z. Li, B. Chen, K. Samel, M. Naik, L. Song, and X. Si. Scallop: From probabilistic deductive databases to scalable differentiable reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 25134–25145, 2021.
37. E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations (ICLR)*, 2017.
38. J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1988–1997, 2017.
39. J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, 2021.

40. H. Kautz. The third AI summer: AAAI robert s. engelmore memorial lecture. *AI Magazine*, 43(1):93–104, 2022.
41. K. Kersting and L. De Raedt. Basic principles of learning bayesian logic programs. *Probabilistic Inductive Logic Programming: Theory and Applications*, pages 189–221, 2008.
42. D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representation (ICLR)*, 2015.
43. T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.
44. D. Kisa, G. Van den Broeck, A. Choi, and A. Darwiche. Probabilistic sentential decision diagrams. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2014.
45. R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L. Li, D. A. Shamma, M. S. Bernstein, and L. Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *Int. J. Comput. Vis.*, 123(1):32–73, 2017.
46. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, 2017.
47. B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40:e253, 2017.
48. M. Law, A. Russo, E. Bertino, K. Broda, and J. Lobo. Fastlas: Scalable inductive logic programming incorporating domain-specific optimisation criteria. *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 2877–2885, 2020.
49. M. Law, A. Russo, and K. Broda. Inductive learning of answer set programs. In *Logics in Artificial Intelligence - 14th European Conference (JELIA)*, 2014.
50. M. Law, A. Russo, and K. Broda. Inductive learning of answer set programs from noisy examples. *Advances in Cognitive Sciences*, 7:57–76, 2018.
51. Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
52. J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning (ICML)*, volume 97, pages 3744–3753, 2019.
53. Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel. Gated graph sequence neural networks. In *International Conference on Learning Representations (ICLR)*, 2016.
54. J. W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, Heidelberg, 1984.
55. F. Locatello, D. Weissenborn, T. Unterthiner, A. Mahendran, G. Heigold, J. Uszkoreit, A. Dosovitskiy, and T. Kipf. Object-centric learning with slot attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
56. C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations (ICLR)*, 2017.
57. M. Malkinski and J. Mandziuk. A review of emerging research directions in abstract visual reasoning. *Information Fusion*, 91:713–736, 2023.
58. R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, and L. De Raedt. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3753–3763, 2018.
59. R. Manhaeve, S. Dumančić, A. Kimmig, T. Demeester, and L. De Raedt. Neural probabilistic logic programming in deepproblog. *Artif. Intell.*, 298:103504, 2021.
60. J. Mao, C. Gan, P. Kohli, J. B. Tenenbaum, and J. Wu. The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision. In *International Conference on Learning Representations (ICLR)*, 2019.
61. S. Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *Proceedings of the 30th Design Automation Conference (DAC)*, pages 272–277, New York, 1993. ACM Press.
62. P. Minervini, S. Riedel, P. Stenetorp, E. Grefenstette, and T. Rocktäschel. Learning reasoning strategies in end-to-end differentiable proving. In *International Conference on Machine Learning, (ICML)*, volume 119, pages 6938–6949, 2020.
63. S. H. Muggleton. Inductive logic programming. *New Gener. Comput.*, 8(4):295–318, 1991.

64. S. H. Muggleton. Inverse entailment and progol. *New Gener. Comput.*, 13:245–286, 1995.
65. S. H. Muggleton, D. Lin, and A. Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: predicate invention revisited. *Mach. Learn.*, 100(1):49–73, 2015.
66. H. Müller and A. Holzinger. Kandinsky patterns. *Artif. Intell.*, 300:103546, 2021.
67. D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia. Efficient large-scale language model training on GPU clusters using megatron-lm. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, New York, 2021. ACM.
68. A. Newell. *Human Problem Solving*. Prentice-Hall, Inc., USA, 1972.
69. A. Newell and H. A. Simon. Computer simulation of human thinking. *Science*, 134(3495):2011–2017, 1961.
70. M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.
71. S.-H. Nienhuys-Cheng, R. d. Wolf, J. Siekmann, and J. G. Carbonell. *Foundations of Inductive Logic Programming*. Springer-Verlag, Berlin, Heidelberg, 1997.
72. M. Nishino, N. Yasuda, S. Minato, and M. Nagata. Zero-suppressed sentential decision diagrams. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 1058–1066, 2016.
73. A. Paliwal, S. M. Loos, M. N. Rabe, K. Bansal, and C. Szegedy. Graph representations for higher-order logic and theorem proving. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 2967–2974, 2020.
74. F. Petersen, C. Borgelt, H. Kuehne, and O. Deussen. Learning with algorithmic supervision via continuous relaxations. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 16520–16531, 2021.
75. G. Plotkin. A further note on inductive generalization. In *Machine Intelligence*, volume 6. Edinburgh University Press, Edinburgh, 1971.
76. H. Poon and P. M. Domingos. Sum-product networks: A new deep architecture. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 337–346, 2011.
77. J. C. Raven and J. H. Court. *Raven's progressive matrices and vocabulary scales*. Oxford Psychologists Press, Oxford, 1998.
78. M. Rawson and G. Reger. Directed graph networks for logical entailment. In *EasyChair Preprint no. 2185*, 2020.
79. J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.
80. M. Richardson and P. M. Domingos. Markov logic networks. *Mach. Learn.*, 62(1-2):107–136, 2006.
81. R. Riegel, A. Gray, F. Luus, N. Khan, N. Makondo, I. Y. Akhalwaya, H. Qian, R. Fagin, F. Barahona, U. Sharma, et al. Logical neural networks. *arXiv Preprint:2006.13155*, 2020.
82. T. Rocktäschel and S. Riedel. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3788–3800, 2017.
83. A. S. Ross, M. C. Hughes, and F. Doshi-Velez. Right for the right reasons: Training differentiable models by constraining their explanations. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2662–2670, 2017.
84. S. Ruder. An overview of gradient descent optimization algorithms. *arXiv Preprint*, abs/2110.09383, 2016.
85. S. J. Russell and P. Norvig. *Artificial Intelligence - A Modern Approach, Third International Edition*. Pearson Education, New York., 2010.
86. T. Sato. A statistical learning method for logic programs with distribution semantics. In *International Conference on Logic Programming (ICLP)*, pages 715–729, 1995.
87. F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.
88. M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *The Semantic Web - 15th International Conference (ESWC)*, volume 10843 of *Lecture Notes in Computer Science*, pages 593–607, 2018.

89. E. Y. Shapiro. *Algorithmic Program DeBugging*. MIT Press, Cambridge, MA, 1983.
90. H. Shindo, M. Nishino, and A. Yamamoto. Differentiable inductive logic programming for structured examples. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 5034–5041, 2021.
91. H. Shindo, V. Pfanschilling, D. S. Dhami, and K. Kersting. αILP: thinking visual scenes as differentiable logic programs. *Mach. Learn.*, 112(5):1465–1497, 2023.
92. D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.
93. T. Silver, K. R. Allen, A. K. Lew, L. P. Kaelbling, and J. Tenenbaum. Few-shot bayesian imitation learning with logical program policies. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 10251–10258, 2020.
94. A. Skryagin, W. Stammer, D. Ochs, D. S. Dhami, and K. Kersting. Neural-probabilistic answer set programming. In *International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 2022.
95. A. Solar-Lezama. *Program Synthesis by Sketching*. PhD thesis, University of California, Berkeley, 2008.
96. E. S. Spelke, K. Breinlinger, J. Macomber, and K. Jacobson. Origins of knowledge. *Psychological review*, 99(4):605, 1992.
97. W. Stammer, P. Schramowski, and K. 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)*, pages 3619–3629, 2021.
98. H. Tan and M. Bansal. LXMERT: learning cross-modality encoder representations from transformers. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5099–5110, 2019.
99. J. B. Tenenbaum, C. Kemp, T. L. Griffiths, and N. D. Goodman. How to grow a mind: Statistics, structure, and abstraction. *Science*, 331(6022):1279–1285, 2011.
100. G. Van den Broeck, K. Kersting, S. Natarajan, and D. Poole. *An Introduction to Lifted Probabilistic Inference*. MIT Press, Cambridge, MA, 2021.
101. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, 2017.
102. G. Šourek, V. Aschenbrenner, F. Železný, S. Schockaert, and O. Kuželka. Lifted relational neural networks: Efficient learning of latent relational structures. *J. Artif. Intell. Res.*, 62:69–100, 2018.
103. L. Weng. How to train really large models on many gpus? *lilianweng.github.io*, Sep 2021.
104. Q. Wu, D. Teney, P. Wang, C. Shen, A. Dick, and A. Van Den Hengel. Visual question answering: A survey of methods and datasets. *Image Vis. Comput.*, 163:21–40, 2017.
105. Z. Yang, A. Ishay, and J. Lee. Neurasp: Embracing neural networks into answer set programming. In C. Bessiere, editor, *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1755–1762, 2020.
106. K. Yi, C. Gan, Y. Li, P. Kohli, J. Wu, A. Torralba, and J. B. Tenenbaum. Clevrer: Collision events for video representation and reasoning. In *International Conference on Learning Representations (ICLR)*, 2020.
107. K. Yi, J. Wu, C. Gan, A. Torralba, P. Kohli, and J. Tenenbaum. Neural-symbolic VQA: disentangling reasoning from vision and language understanding. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1039–1050, 2018.
108. S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim. Graph transformer networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 11960–11970, 2019.
109. C. Zhao, C. Xiong, C. Rosset, X. Song, P. Bennett, and S. Tiwary. Transformer-xh: Multi-evidence reasoning with extra hop attention. In *International Conference on Learning Representations (ICLR)*, 2020.

A Logic Programs for Behind-the-Scenes

We show logic programs used for solving the Behind-the-Scenes task but not shown in the main text. Fig. 18 shows a set of clauses to define utility predicates to solve the task, *e.g.* extracting a color of an object according to its position. Fig. 19 shows additional logic programs used for query answering.

```
% extract the target color according to the position
% e.g. get the color of the 1st (left-most) object
get_color([Color1,Color2,Color3],Position,X):-
              first_obj([Color1,Color2,Color3],X),
              same_position(Position,1st).
get_color([Color1,Color2,Color3]),Position,X):-
              second_obj([Color1,Color2,Color3]),X),
              same_position(Position,2nd).
get_color([Color1,Color2,Color3],Position,X):-
              third_obj([Color1,Color2,Color3],X),
              same_position(Position,3rd).

% mapping the list of colors to a color, according to the position
% used by the get_color predicate
first_obj([Color1,Color2,Color3],Color1):-.
second_obj([Color1,Color2,Color3],Color2):-.
third_obj([Color1,Color2,Color3],Color3):-.
```

Fig. 18 Additional clauses to define utility predicates to answer queries.

```
% append(X,Y,Z) means Z is the result of appending X to the head of Y.
append([],X,X):-.
append([X|Y],Z,[X|V]):-append(Y,Z,V).

% reverse(X,Y) means that Y is the result of reversing X.
reverse([H|T],A,R):-reverse(T,[H|A],R).
reverse([],A,A):-.

% permutation(X,Y) means Y is a permutation of X.
permutation([X,Y],Z):-permutation(Y,V),delete(X,Z,V).
```

Fig. 19 Additional clauses to compute abstract list operations.

B Experimental Details on Reasoning Behind-the-Scenes

We describe the experimental details of the behind-the-scenes task.

Task 1 (T1). We used the mode declarations in Tab. 8 for both NEUMANN and ∂ILP-ST. The definition of mode declarations is available in Sec. D. We performed 50 epochs of weight optimization with a batch size of 64. We used the RMSProp [84] optimizer with

Table 8 Mode declarations for Behind-the-Scenes.

`modeb(1, member(+object, +objcets))`
`modeb(1, delete(+object, +objcets, +objects))`
`modeb(1, is_sorted(+objects))`
`modeb(1, smaller(+object, +object))`
`modeb(1, permutation(+objects, +objcets))`

a learning rate of $1e-2$. To prune too general clauses in the clause generation step, we gradually increased the ratio of negative examples from 20% to 100% by 20% in the first 5 trials of the clause generation. For `sort`, we performed curriculum learning, *e.g.* learning a simple predicate first (`is_sorted`) and then finalizing the complete learning (`sort`).

We limit the number of nested functors at most 3 and discard lists with duplicated elements. We used the same perception model as in CLEVR-Hans, which is described in Section C.3. Visual examples for each operation are shown in Sec. E.

Task 2 (T2). Queries about different operations are given randomly, so the model needs to handle different types of queries in the prediction. To generate visual scenes with queries and their answers, we adopted the generation code of CLEVR [38]. When solving, NEUMANN reads out a JSON file which contains instances, and each instance contains a path to an image file and pairs of a query and a corresponding answer, *e.g.* (`query2(q_delete, cyan, 2nd)`, `ans(red)`). NEUMANN assigns probabilities over query atoms according to the input, *i.e.* it gives 1.0 for a query atom given input and 0.0 for other query atoms. The answer is used only for computing accuracies and never in the prediction pipeline.

We used a batch size of 64 for NEUMANN and 1 for αILP, *i.e.* αILP produced out-of-memory even with the smallest batch size, as shown in Tab. 6. We used the same perception model as in CLEVR-Hans, which is described in Section C.3. We limit the number of nested functors at most 3 and discard lists with duplicated elements. We used additional abstract operations (`append` and `reverse`) shown in Fig. 19. Examples of input scenes paired with queries and their answers are shown in Sec. E.

C Experimental Details on Kandinsky Patterns and CLEVR-Hans

In this section, we describe the experimental setting of Kandinsky Patterns and CLEVR-Hans.

C.1 Kandinsky Patterns

CNN. We trained ResNet18 for 300 epochs with a batch size of 512. We used the Adam optimizer [42, 84] with a learning rate of $1e-5$.

YOLO+MLP. We used MLP with *two* hidden layers. Each hidden layer applies a linear transformation and a non-linearity. The output of the pre-trained YOLO model is reshaped and fed into MLP to predict the class label. We jointly trained the whole YOLO+MLP network for 1000 epochs with a batch size of 512. We used the Adam optimizer [42, 84] with a learning rate of $1e-5$.

αILP/NEUMANN. We trained the αILP and NEUMANN model for 100 epochs with a batch size of 64. We used the RMSProp [84] optimizer with a learning rate of $1e-2$. For αILP, we used 500 positive examples in the validation split to generate clauses by beam search.

Mode declarations we used are shown in Tab. 9. Tab. 10 shows the data types and constants, and Tab. 11 shows the predicates used in our experiments. #obj represents the number of objects to be focused on the classification, which can be identified by trying from the smallest number and evaluating by validation split and increasing if the performance is not enough. We set the initial clause to be the root node in the beam search as:

Table 9 Mode declarations for Kandinsky patterns and CLEVR-Hans.

modeh(1, kp(−image))
modeb(#obj, in(−object, +image))
modeb(1, color(+object, #color))
modeb(1, shape(+object, #shape))
modeb(2, same_color_pair(+object, +object))
modeb(2, same_shape_pair(+object, +object))
modeb(1, diff_color_pair(+object, +object))
modeb(1, duff_shape_pair(+object, +object))
modeb(1, closeby(+object, +object))
modeb(1, online(+object, . . . , +object))
modeh(1, ch(−image))
modeb(#obj, in(−object, +image))
modeb(1, color(+object, #color))
modeb(1, shape(+object, #shape))
modeb(1, material(+object, #material))
modeb(1, size(+object, #size))

Table 10 Datatype and constants in Kandinsky patterns and CLEVR-Hans.

Datatype	Terms
image	img
object	obj1, obj2, . . ., obj6
color	red, blue, yellow
shape	square, circle, triangle
image	img
object	obj0, obj1, . . ., obj9
color	cyan, blue, yellow, purple, red, green, gray, brown
shape	sphere, cube, cylinder
size	large, small
material	rubber, metal

Table 11 Predicates in the Kandinsky patterns and CLEVR-Hans.

Predicate	Explanation
kp/(1, [image])	The image belongs to the Kandinsky pattern.
same_shape_pair/(2, [object, object])	The two objects have the same shape.
same_color_pair/(2, [object, object])	The two objects have the same color.
diff_shape_pair/(2, [object, object])	The two objects have different shapes.
diff_color_pair/(2, [object, object])	The two objects have different colors.
in/(2, [object, image])	The object is in the image.
shape/(2, [object, shape])	The object has the shape of the second argument.
color/(2, [object, color])	The object has the color of the second argument.
closeby/(2, [object, object])	The two objects are located close by each other.
online/(5, [object, . . . , object])	The objects are aligned on a line.
ch/(1, [image])	The image belongs to the clevr-hans pattern.
in/(2, [object, image])	The object is in the image.
shape/(2, [object, shape])	The object has the shape of the second argument.
color/(2, [object, color])	The object has the color of the second argument.
material/(2, [object, material])	The object has the material of material.
size/(2, [object, size])	The object has the size of the second argument.

kp(X) :- in(O1, X), . . . , in(On, X)., where n is the number of objects to be focused. Background knowledge given in for Kandinsky patterns is shown in Tab. 12.

C.2 CLEVR-Hans

We trained the αILP and NEUMANN model for 100 epochs with a batch size of 256. We used the RMSProp optimizer [84] with a learning rate of $1e-2$. For αILP, we used 500 positive examples in the validation split to generate clauses by beam search.

Mode declarations we used are shown in Tab. 9. Tab. 10 shows the data types and constants, and Tab. 11 shows the predicates. We set the initial clause to be the root node in the beam search as: ch(X) :- in(O1, X), in(O2, X). We did not provide any background knowledge for CLEVR-Hans tasks. We refer to [97] for details about CNN, NeSy, and NeSy-XIL benchmarks.

same_shape_pair(X, Y) : −shape(X, Z), shape(Y, Z),
same_color_pair(X, Y) : −color(X, Z), color(Y, Z),
diff_shape_pair(X, Y) : −shape(X, Z), shape(Y, W), diff_shape(Z, W).
diff_color_pair(X, Y) : −color(X, Z), color(Y, W), diff_color(Z, W),
diff_color(red, blue), diff_color(blue, red),
diff_color(red, yellow), diff_color(yellow, red),
diff_color(blue, yellow), diff_color(yellow, blue).
diff_shape(circle, square), diff_shape(square, circle),
diff_shape(circle, triangle), diff_shape(triangle, circle),
diff_shape(square, triangle), diff_shape(triangle, square).

Table 12 Background knowledge for Kandinsky patterns.

C.3 Perception Models

We describe the experimental setting of the pre-training of the perception models in our experiments.

C.3.1 YOLO for Kandinsky Patterns

Model. We used YOLOv5[7] model, whose implementation is publicly available. We adopted the YOLOv5s model, which has 7.3M parameters.
Dataset. We generated 15,000 pattern-free figures for training, 5000 figures for validation. The class labels and positions are generated randomly. The original image size is 620×620, and resized into 128×128. The label consists of the class labels and the bounding box for each object. The class label is generated by the combination of the shape and the color of the object, e.g., *red circle* and *blue square*. The number of classes is 9. Each image contains at least 2 objects and, at most 10 objects.
Optimization. We trained the YOLOv5s model by stochastic gradient descent (SGD) for 400 epochs using the pre-trained weights[8]. We used the loss function that approximates detection performance, presented in [79]. We set the learning rate to 0.01 and the batch size to 64. The SGD optimizer used the momentum, which is set to 0.937. We set the weight decay as 0.0005. We took 3 warmup epochs for training.

C.3.2 Slot Attention for CLEVR

We used the same model and training setup as the pre-training of the slot-attention module in [97]. In the preprocessing, we downscaled the CLEVR images to a dimension of 128×128 and normalized the images to lie between -1 and 1. For training the slot-attention module, an object is represented as a vector of binary values for the shape, size, color, and material attributes and continuous values between 0 and 1 for the x, y, and z positions. We trained the slot attention model with the set prediction architecture following [55], using the loss function, which is based on the Hungarian algorithm. We refer to [97] for more details.

D Mode Declaration

Mode Declaration [64] is one of the common language biases for Inductive Logic Programming. We used mode declaration, which is defined as follows. A mode declaration is either a head declaration $\texttt{modeh}(\texttt{r}, \texttt{p}(\texttt{mdt}_1, \ldots, \texttt{mdt}_\texttt{n}))$ or a body declaration $\texttt{modeb}(\texttt{r}, \texttt{p}(\texttt{mdt}_1, \ldots, \texttt{mdt}_\texttt{n}))$, where $\texttt{r} \in \mathbb{N}$ is an integer, $\texttt{p}$ is a predicate, and $\texttt{mdt}_\texttt{i}$ is a mode datatype. A mode datatype is

[7] https://github.com/ultralytics/yolov5
[8] https://github.com/ultralytics/yolov5/releases

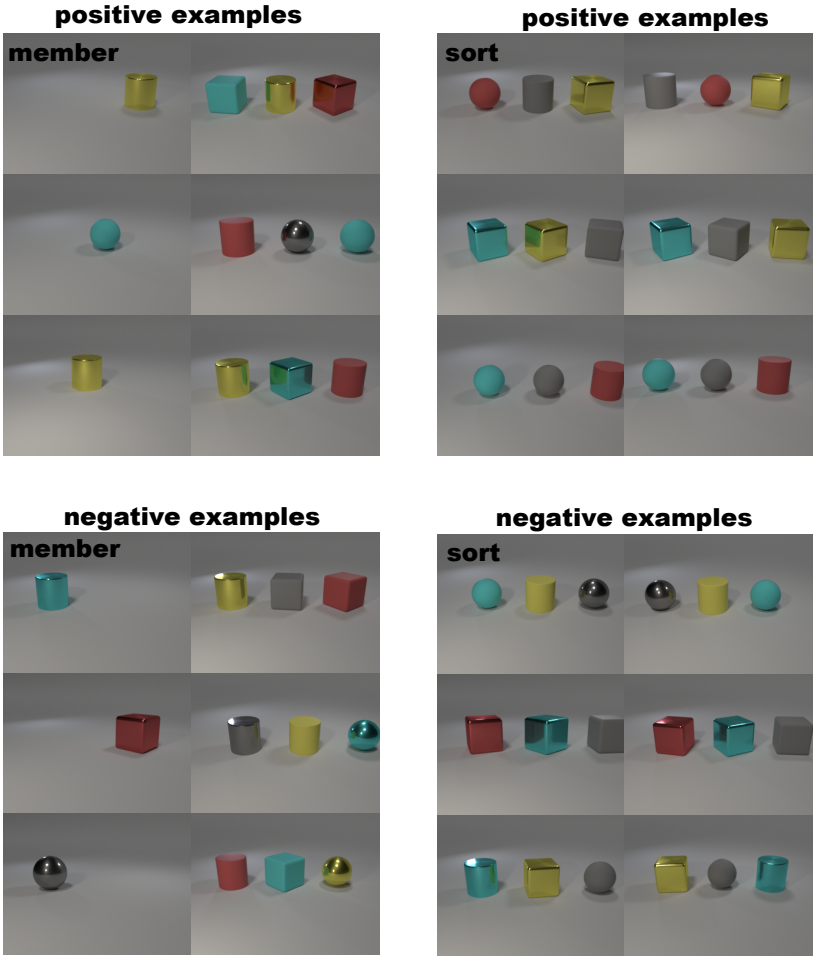


Fig. 20 Positive and negative examples for *member* and *sort* (with an order of colors: *cyan* $<$ *gray* $<$ *red* $<$ *yellow*, alphabetical order) in CLEVR-List.

a tuple $(\texttt{pm}, \texttt{dt})$, where `pm` is a place-marker and `dt` is a datatype. A place-marker is either $\#$, which represents constants, or $+$ (resp. $-$), which represents input (resp. output) variables. `r` represents the number of the usages of the predicate to compose a single clause.

E More Examples in Datasets

Fig. 20 shows some positive and negative examples for *member* and *sort* in CLEVR-List. Fig. 21 shows some positive and negative examples for *delete* in CLEVR-List. We show some examples of visual input, queries, and their answers in the behind-the-scenes task in Fig. 22. We show some examples for each pattern we used in Kandinsky patterns in Fig. 23. We also show some examples for each class of CLEVR-Hans in Fig. 24.

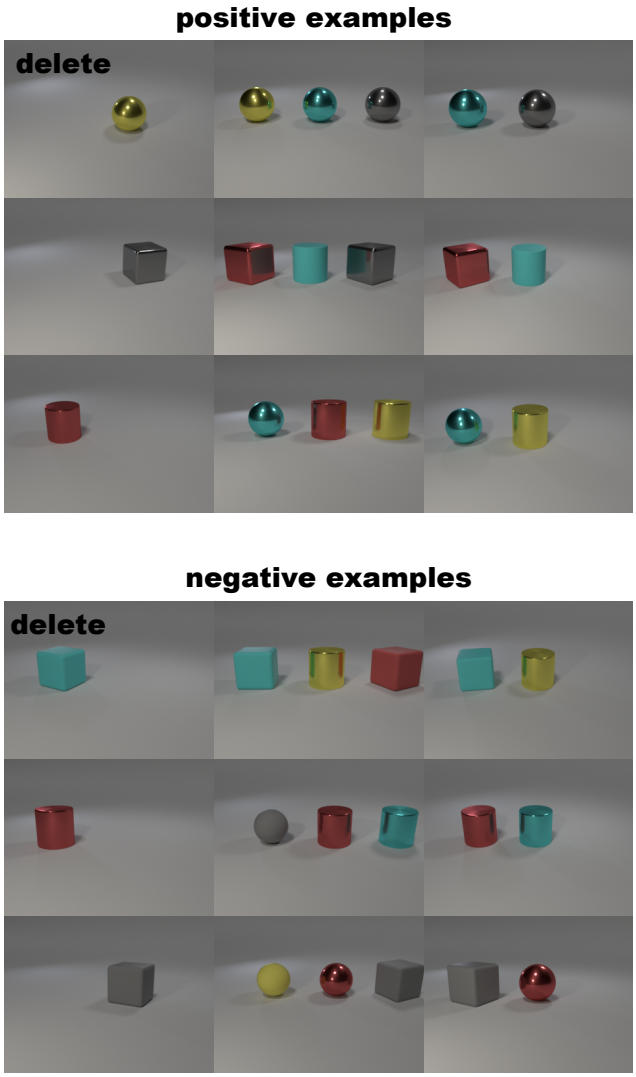
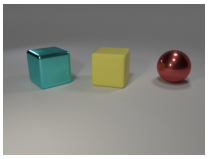


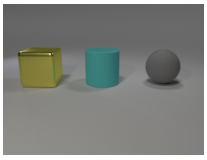
Fig. 21 Positive and negative examples for *delete* in CLEVR-List.

Image



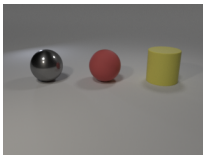
Query	Answer
query3(delete,cyan,2nd)	ans(red)
query3(append,gray,2nd)	ans(cyan)
query2(reverse,1st)	ans(red)
query2(sort,3rd)	ans(yellow)

Image



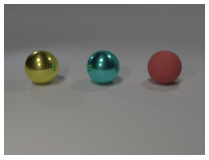
Query	Answer
query3(delete,cyan,2nd)	ans(gray)
query3(append,red,1st)	ans(red)
query2(reverse,3rd)	ans(yellow)
query2(sort,2nd)	ans(gray)

Image



Query	Answer
query3(delete,gray,2nd)	ans(yellow)
query3(append,cyan,2nd)	ans(gray)
query2(reverse,1st)	ans(yellow)
query2(sort,3rd)	ans(yellow)

Image

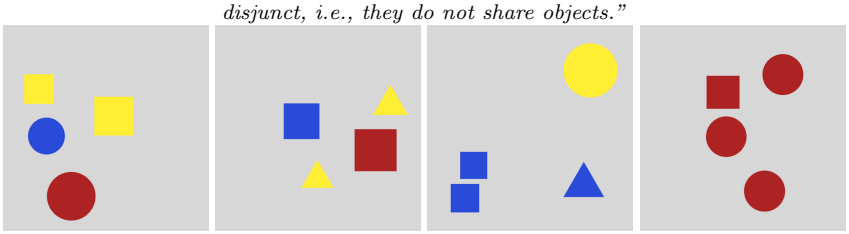


Query	Answer
query3(delete,cyan,2nd)	ans(red)
query3(append,gray,1st)	ans(gray)
query2(reverse,3rd)	ans(yellow)
query2(sort,2nd)	ans(red)

Fig. 22 Examples of visual input, queries, and their answers in the Behind-the-Scenes task. Input is given as a pair of a visual scene and a query.

TwoPairs

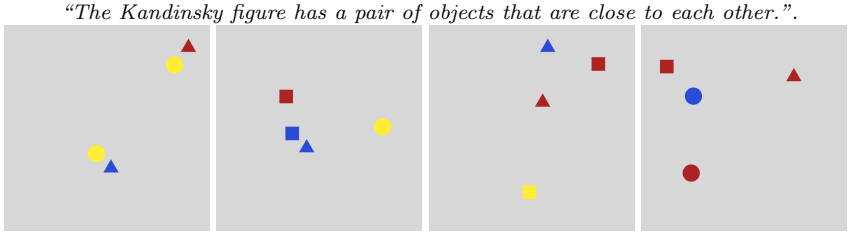
"The Kandinsky figure has two pairs of objects with the same shape. In one pair, the objects have the same colors in the other pair different colors. Two pairs are always disjunct, i.e., they do not share objects."



positive positive negative negative

Closeby

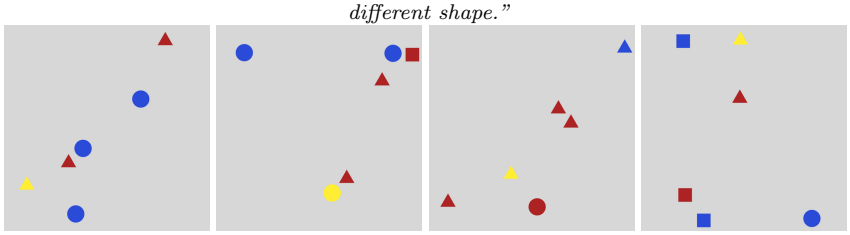
"The Kandinsky figure has a pair of objects that are close to each other.".



positive positive negative negative

Red-Triangle

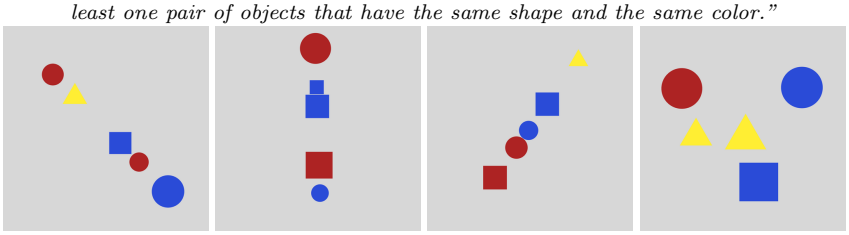
"The Kandinsky figure has a pair of objects that are close to each other, and one object of the pair is a red triangle, and the other object has a different color and different shape. (A red triangle is attacking someone who has a different color and a different shape."



positive positive negative negative

Online/Pair

"The Kandinsky figure has five objects that are aligned on a line, and it contains at least one pair of objects that have the same shape and the same color."



positive positive negative negative

Fig. 23 Examples in each Kandinsky pattern in our experiments. The left two images are positive examples, and the right two images are negative examples.

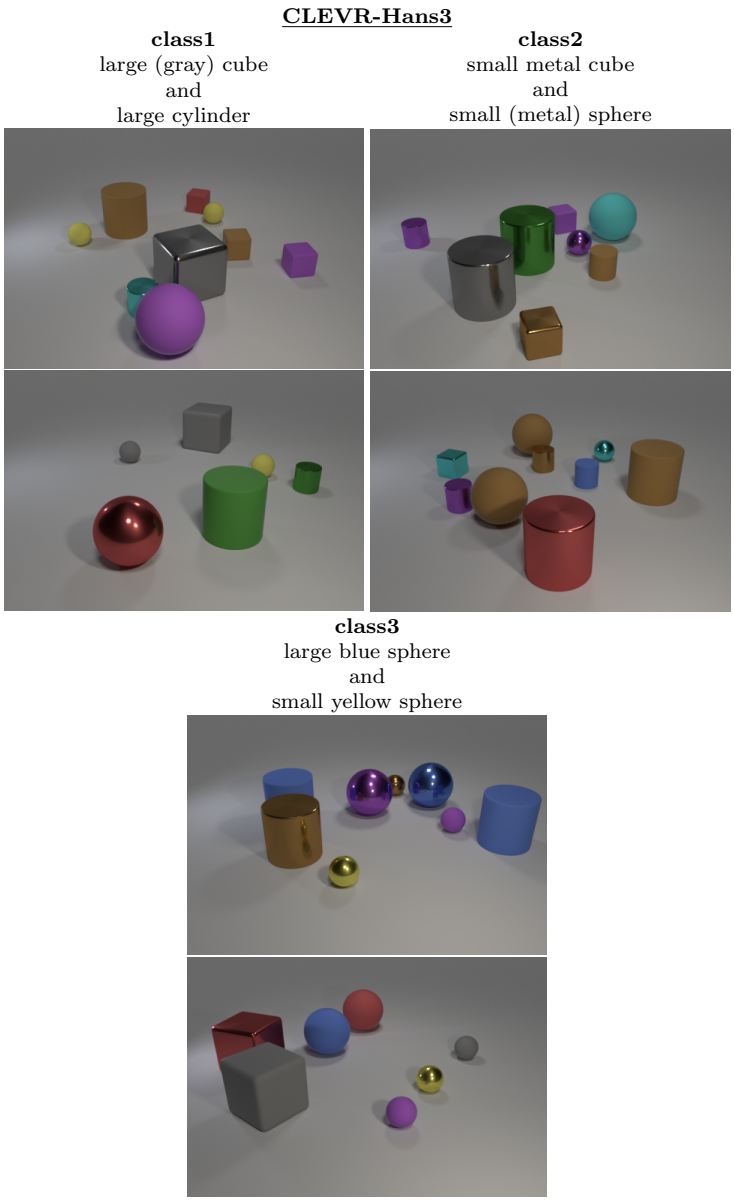


Fig. 24 Examples in CLEVR-Hans3 dataset [97]. The dataset consists of *three* classes. Two images are shown for each class. The text on the top of the images describes the confounded classification rule for each class. For example, images of the first class contain a large cube and a large cylinder. The large cube has the color of gray in every image of the train and validation split. In the test split, the color of the large cube is shuffled randomly.