Neuro-Symbolic Predicate Invention: Learning Relational Concepts from Visual Scenes

Jingyuan Sha [a,*], Hikaru Shindo [a], Kristian Kersting [a,b,c] and Devendra Singh Dhami [d]
[a] *Computer Science Department and Centre for Cognitive Science, Technische Universität Darmstadt, Germany*
[b] *Hessian Center for Artificial Intelligence (hessian.AI), Germany*
[c] *German Research Centre for Artificial Intelligence (DFKI), Germany*
[d] *Department of Mathematics and Computer Science, Eindhoven University of Technology, Netherlands*

Abstract. The predicates used for Inductive Logic Programming (ILP) systems are usually elusive and need to be hand-crafted in advance, which limits the generalization of the system when learning new rules without sufficient background knowledge. Predicate Invention (PI) for ILP is the problem of discovering new concepts that describe hidden relationships in the domain. PI can mitigate the generalization problem for ILP by inferring new concepts, giving the system a better vocabulary to compose logic rules. Although there are several PI approaches for symbolic ILP systems, PI for Neuro-Symbolic-ILP (NeSy-ILP) systems that can handle 3D visual inputs to learn logical rules using differentiable reasoning is still unaddressed. To this end, we propose a neuro-symbolic approach, NeSy-π, to invent predicates from visual scenes for NeSy-ILP systems based on clustering and extension of relational concepts, where π denotes the abbrivation of **P**redicate **I**nvention. NeSy-π processes visual scenes as input using deep neural networks for the visual perception and invents new concepts that support the task of classifying complex visual scenes. The invented concepts can be used by any NeSy-ILP system instead of hand-crafted background knowledge. Our experiments show that the NeSy-π is capable of inventing high-level concepts and solving complex visual logic patterns efficiently and accurately in the absence of explicit background knowledge. Moreover, the invented concepts are explainable and interpretable, while also providing competitive results with state-of-the-art NeSy-ILP systems. (github: https://github.com/ml-research/NeSy-PI)

Keywords: Predicate Invention, Inductive Logic Programming, Neuro-Symbolic Artificial Intelligence

1. Introduction

Inductive Logic Programming (ILP) learns generalized logic programs from data [1–3]. Unlike Deep Neural Networks (DNNs), ILP gains vital benefits, including the capacity to discover explanatory rules from a small number of examples. Nevertheless, predicates for ILP systems are typically elusive and need to be hand-crafted, requiring additional prior knowledge to compose solutions. This makes the wide-scale adaption of such systems difficult and taxing. Predicate invention (PI) systems invent new predicates that map new concepts from expert-designed primitive predicates. This extends the expression of the ILP language and consequently decreases reliance on human experts [4]. A simple example is the concept of the *blue sphere*, which is the combination of two primitive concepts *blue* and *sphere*. Consider the visual scenes illustrated in Fig. 1, where we see several objects in the scene, and

[*]Corresponding author. E-mail: jingyuan.sha@tu-darmstadt.de.

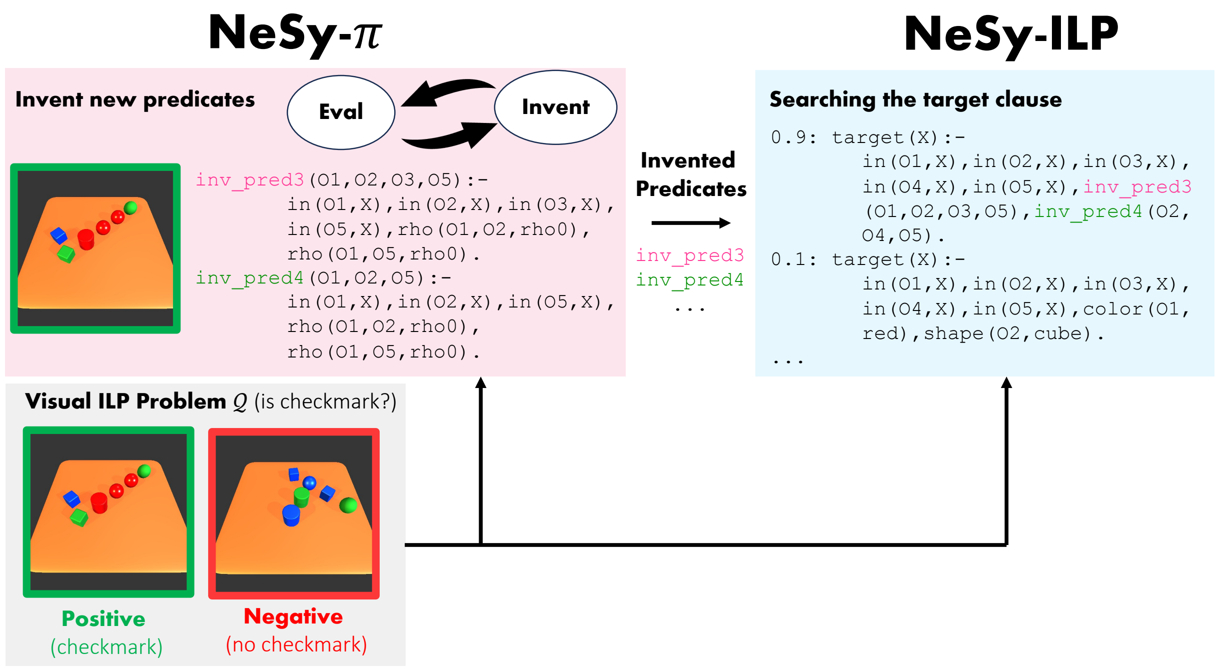

NeSy-π
NeSy-ILP
Invent new predicates
Eval
Invent
inv_pred3(O1,O2,O3,O5):- in(O1,X),in(O2,X),in(O3,X), in(O5,X),rho(O1,O2,rho0), rho(O1,O5,rho0).
inv_pred4(O1,O2,O5):- in(O1,X),in(O2,X),in(O5,X), rho(O1,O2,rho0), rho(O1,O5,rho0).
Invented Predicates
inv_pred3
inv_pred4
...
Searching the target clause
0.9: target(X):- in(O1,X),in(O2,X),in(O3,X), in(O4,X),in(O5,X),inv_pred3 (O1,O2,O3,O5),inv_pred4(O2, O4,O5).
0.1: target(X):- in(O1,X),in(O2,X),in(O3,X), in(O4,X),in(O5,X),color(O1, red),shape(O2,cube).
...
Visual ILP Problem Q (is checkmark?)
Positive
(checkmark)
Negative
(no checkmark)

Fig. 1. **NeSy-π discovers relational concepts from visual scenes.** NeSy-π develops relational concepts for visual scenes through iterative *evaluation* and *invention* of primitive predicates and visual scenes (on the left). The invented predicates are fed into a NeSy-ILP solver to learn classification rules. The NeSy-ILP solver creates rule candidates utilizing given predicates and performs differentiable reasoning and learning (on the right). Consequently, the classification rules can be effectively constructed with the invented predicates (best viewed in color).

the task is to reason and learn about objects' attributes and their relations. With PI systems, the non-primitive concepts (e.g. *blue sphere*) are not explained in the background knowledge, but have to be learned from data. For instance, given training data, a PI system can invent new predicate `blue_sphere` defined by the following rule: `blue_sphere(O):-color(O,blue),shape(O,sphere)`, where `O` is a variable, `blue` and `sphere` are constants, and `color` and `sphere` are primitive predicates provided by experts. By using the invented predicate `blue_sphere`, ILP systems can compose rules efficiently to solve problems.

Recently, a differentiable ILP framework has been proposed to integrate symbolic ILP with DNNs [5] solving rule-learning tasks on symbolic or simple visual inputs, *e.g.* hand-written images. In a similar vein, neuro-symbolic ILP (NeSy-ILP) systems [1] that can learn explanatory rules on complex visual scenes (*e.g.* CLEVR scenes [7]) have shown their capability to reason and learn on abstract concepts and relations solving complex visual patterns [6]. NeSy-ILP systems outperform pure neural baselines on complex visual reasoning, since it involves inferring answers by reasoning about objects' attributes and their relations [8]. The main drawback of the current NeSy-ILP systems is that they necessitate all predicates in advance, such as pre-trained neural predicates or manually constructed background knowledge. In addition, obtaining background knowledge is expensive as it requires human experts or the pre-training of neural modules with supplementary supervision. Consequently, this significantly restricts the flexibility of the NeSy-ILP systems across various domains. Therefore, a question arises: *How can a NeSy-ILP system learn with less or no background knowledge in complex visual scenes?*

To address this problem, we introduce *Neuro-Symbolic Predicate Invention (NeSy-π)*, which can invent relational concepts from visual scenes without additional supervision or hard coding. NeSy-π provides a rich vocabulary for NeSy-ILP systems to produce effective solutions on 3D visual scenes. A briefly applying case is illustrated in Fig. 1.

[1]There have been proposed different instances of NeSy-ILP systems to integrate DNNs with ILP. In this work, we particularly focus on αILP [6], which performs structure learning on given positive and negative visual scenes, where each visual scene is converted to a set of probabilistic atoms, and each weighted rule describes the scenes.

Given positive and negative examples as visual scenes, NeSy-π performs PI finding useful relational concepts, *e.g.* grouping several objects and capturing their spatial relations to compose complex patterns. Given primitive predicates, NeSy-π performs the following 2 steps iteratively: **(Step 1: Eval)** evaluating available predicates invented in the previous steps, and **(Step 2: Invent)** inventing new predicates based on the evaluation scores. Additionally, we propose three novel metrics to evaluate invented predicates resulting in pruning of the redundant candidates efficiently and gain high-quality vocabulary. The invented predicates can then be fed to NeSy-ILP systems to learn explanatory classification rules.

Overall, we make the following important contributions:

1. We propose NeSy-π, a novel neuro-symbolic predicate invention framework compatible with NeSy-ILP systems. It extends NeSy-ILP systems by providing the capability of learning vocabularies from 3D visual scenes. NeSy-π enables them to learn using less background knowledge from human experts, mitigating the scaling bottleneck of the current NeSy-ILP systems.
2. To evaluate NeSy-π, we propose three metrics. These metrics measure the percentage of examples covered or eliminated by the predicates, enabling NeSy-π to efficiently discovers useful concepts.
3. To evaluate the ability of predicate invention using visual inputs, we propose 3*D Kandinsky Patterns*, extending Kandinsky Patterns [9] to the 3D world. The Kandinsky Patterns environment can generate positive and negative visual scenes using different abstract patterns. However, it has been limited to simple 2D images, and predicate invention has not been addressed. Thus we propose the first environment for the evaluation of neuro-symbolic predicate invention systems that can process complex visual scenes, filling the gap between the abstract synthetic tasks and realistic 3D environments.
4. We empirically show that NeSy-π solves challenging visual reasoning tasks outperforming the conventional NeSy-ILP systems without predicate invention. In our experiments, we successfully applied NeSy-π to 3D Kandinsky Patterns to invent new predicates for complex scenes, achieving higher performances than baselines. Moreover, NeSy-π produces highly-interpretable rules using invented predicates, which cannot be learned by the previous systems.

We have made our code publicly available at https://github.com/ml-research/NeSy-PI. We proceed as follows: we present the required background knowledge before introducing our NeSy-π architecture. We then illustrate the effectiveness of our approach using extensive experiments. Finally, we discuss the related work before concluding.

2. First-order Logic and Inductive Logic Programming

Before introducing NeSy-π, we revisit the basic concepts of first-order logic and Inductive Logic Programming **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 consisting of a functor. A *ground term* is a term with no variables. We denote an n-ary predicate $\mathtt{p}$ by $\mathtt{p}/n$. An *atom* is a formula $\mathtt{p}(\mathtt{t}_1, \ldots, \mathtt{t}_\mathtt{n})$, where $\mathtt{p}$ is an n-ary predicate symbol and $\mathtt{t}_1, \ldots, \mathtt{t}_\mathtt{n}$ 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 simply an atom. A *negative literal* is the negation of an atom. A *clause* is a finite disjunction ($\lor$) 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 $A \lor \neg B_1 \lor \ldots \lor \neg B_n$ is a definite clause. We write definite clauses in the form of A :- $B_1, \ldots, B_n$. The atom A is called the *head*, and the set of negative atoms $\{B_1, \ldots, B_n\}$ is called the *body*. For simplicity, we refer to the definite clauses as clauses in this paper.

Inductive Logic Programming (ILP). An ILP problem $\mathcal{Q}$ is a tuple $(\mathcal{D}^+, \mathcal{D}^-, \mathcal{B}, \mathcal{L})$, where $\mathcal{D}^+$ is a set of positive examples, $\mathcal{D}^-$ is a set of negative examples, $\mathcal{B}$ is background knowledge, and $\mathcal{L}$ is a language. We assume that the examples and background knowledge are ground atoms. The solution of an ILP problem is a set of definite clauses $\mathcal{H} \subseteq \mathcal{L}$ that satisfies the following conditions: (1) $\forall A \in \mathcal{D}^+,\ \mathcal{H} \cup \mathcal{B} \models A$ and (2) $\forall A \in \mathcal{D}^-,\ \mathcal{H} \cup \mathcal{B} \not\models A$. Typically, a search algorithm starts with general clauses and incrementally weakens them through the process known as *refinement*. Refinement is a crucial tool for ILP when the current state clauses are overly general and result in too many negative examples being included.

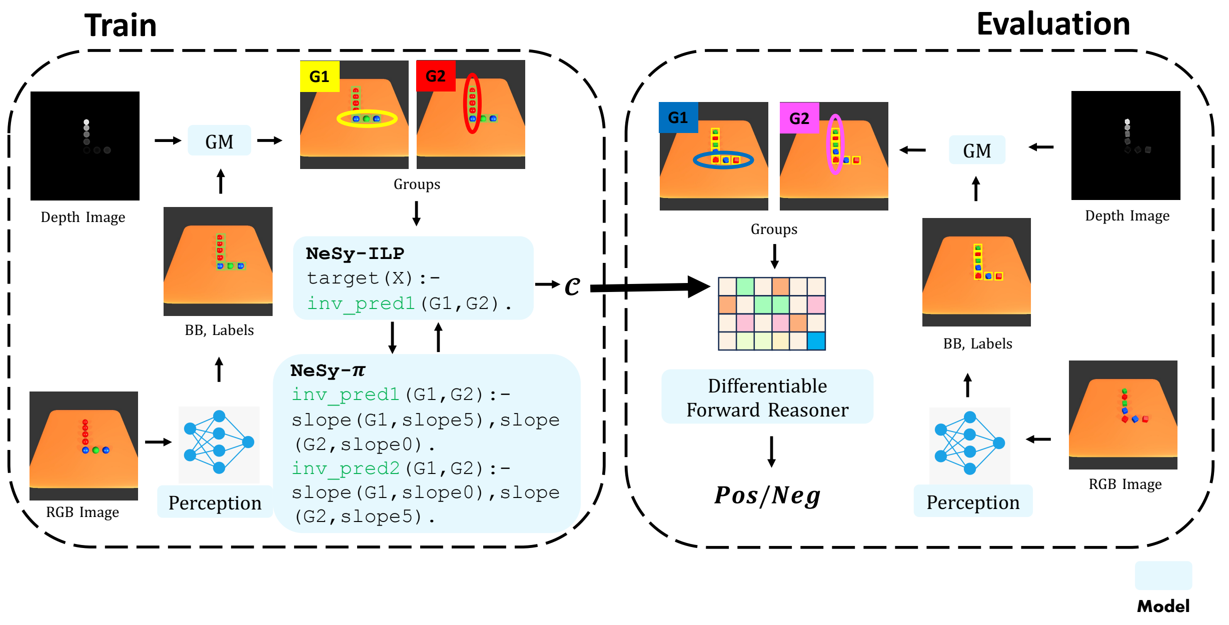


Fig. 2. **Workflow of Visual Scene Reasoning (Train, Left)** During the training phase, NeSy-π learns a set of rules $\mathcal{C}$ to describe common patterns within the given RGB-D images. This process involves using four models: A pretrained perception module (e.g. Mask R-CNN [10]) for object detection; A grouping module (GM) that fits objects to lines base on their 3D positions; NeSy-ILP as a rule learner; NeSy-π as a concept inventor. GM groups objects together (*e.g.* a group of objects aligned on a line) to form complex spatial relations (*e.g.* a check mark). **(Evaluate, Right)** In the evaluation phase, given a test RGB-D image, the system predicts whether the image follows the set of rules $\mathcal{C}$ (positive) or not (negative). The GM and perception module are used for image processing, while a differentiable forward reasoner is employed for reasoning purposes.

NeSy Inductive Logic Programming. We address the problem of Neuro-Symbolic Inductive Logic Programming (NeSy-ILP) presented in 3D visual scenes, which we refer to as the "Visual ILP Problem". The classification pattern is based on high-level concepts encompassing attributes and object relations. To solve visual ILP problems, differentiable ILP frameworks have been proposed such as ∂ILP [5] and αILP [6]. They employ a differentiable implementation of *forward reasoning* in first-order logic(FOL) to enable gradient-based learning of the classification rules. The optimization problem they solve to this end is $\min_{\mathcal{W}} loss(\mathcal{Q}, \mathcal{C}, \mathcal{W})$, whereby $\mathcal{Q}$ is an ILP problem, $\mathcal{C}$ represents the set of clause candidates, $\mathcal{W}$ denote the set of clause weights, and *loss* is a loss function that returns a penalty when training constraints are breached.

3. Neuro-Symbolic Predicate Invention: NeSy-π

In this section, we present the predicate invention system NeSy-π in the following steps: Section 3.1 is an overview of the system. Section 3.2 explains the clause extension, where the extended clauses are one of the inputs of the NeSy-π system. Section 3.3 explains the clause evaluation, which employs two metric proposed in this paper. The scores assigned to the clauses form the other input of the NeSy-π system. Section 3.4 presents the predicate invention and its evaluation. Section 3.5 presents the grouping module, a data prepossessing module that combines related objects into groups to simplify the problem. Section 3.6 provides the pseudo code for the system.

3.1. Architecture Overview

The NeSy-π is a neuro-symbolic system to invent high-level relational concepts as predicates. The invented predicates are further be used by NeSy-ILP system for target clause searching. The workflow of solving visual

ILP problem using the NeSy-π system is illustrated in Fig. 2. The process is divided into two major components: the training and the evaluation. Training utilizes the training examples $\mathcal{D} = \{\mathcal{D}^+, \mathcal{D}^-\}$ to search for target clauses $\mathcal{C}$, which satisfies two conditions: 1) $\forall A \in \mathcal{D}^+, \mathcal{C} \cup \mathcal{B} \models A$ and 2) $\forall A \in \mathcal{D}^-, \mathcal{C} \cup \mathcal{B} \not\models A$, as the target clauses and selected background knowledge should entail only positive images and no negative images. During evaluation, test examples will be classified as positive or negative based on the target clauses $\mathcal{C}$ returned during the training.

As a prepossessing step, we prepared a dataset for training a perception module, such as Mask R-CNN [10]. The dataset for perception module is similar as training examples in $\mathcal{D}$, which includes scenes with same object types but positioned randomly. Each scene is annotated with object labels and bounding boxes.

Once the perception module is trained, the four modules, perception module, grouping module, NeSy-ILP, and NeSy-π perform an end-to-end training, which takes training examples $d \in \mathcal{D}$ as input and reasons target clauses $\mathcal{C}$. Firstly, the perception module takes the RGB image of training example d to detect the labels and the bounding boxes of the objects, then the 3D positions of each object can then be calculated by utilizing depth map of each example d, the camera matrix K along with the labels and bounding boxes from the output of perception module. After acquiring the object positions, for providing simple and high-level descriptions of the scenes, the *Grouping Module (GM)* combines objects into groups based on their positions and encodes these scenes as *group tensors*, which capture the geometric information of the group, such as dimension scales and positions. These group tensors are then utilized as input for the NeSy-ILP system, which iteratively generates and evaluates clauses on the group tensors to identify the target clauses $\mathcal{C}$. In each iteration of clause searching, NeSy-ILP outputs the searched clauses and their respective evaluation scores to NeSy-π. NeSy-π then invents new predicates by taking disjunctions of the promising clauses and returns them to NeSy-ILP. By incorporating these new invented predicates, NeSy-ILP can use them in the subsequent iterations to search for new clauses with greater precision in describing the rules. The final output of the training session is the set of target clauses produced by the NeSy-ILP system.

During evaluation, the system classifies the test scenes as either positive or negative. Positive scenes satisfy the target clauses while negative scenes do not. The perception and grouping module first interpret the data into group tensors, just as during training. Then, a differentiable forward reasoner is used to classify the group tensors as positive or negative based on the target clauses $\mathcal{C}$.

3.2. *Clause Extension*

We take the *top-down* approach for ILP, where new clauses are generated by specifying general rules [2]. We extend clauses step by step by evaluating on visual scenes to prune redundant candidates. NeSy-π begins with the most general clause for the search, *e.g.* the *initial clause* C_0 is defined as follows:

```
(C_0) target(X):-in(O1,X).
```

where the atom `in(O1,X)` consists of the 2-arity predicate `in/2` and variables `O1,X`, which represents the confidence of the object `O1` in the image `X` detected by the perception module. The atom `target(X)` provides the probability that the image `X` is positive based on the atoms in the clause body. C_0 is the most general clause that assesses the object's existence in the image without imposing any property constraints, such as color or shape.

The clauses set is extended incrementally, starting with the extension of C_0 in the first step. Then a set of clauses $\mathcal{C} = \{C_1, ...C_n\}$ is generated where each new clause extends the previous clause by an atom from the language $\mathcal{L}$ [2]. For instance, by adding atoms `color(O1,red)` and `color(O1,blue)` to the body of C_0, the clause set $\mathcal{C} = \{C_1, C_2\}$ is generated as follows,

```
(C_1) target(X):-in(O1,X),color(O1,red).
(C_2) target(X):-in(O1,X),color(O1,blue).
```

The clauses in $\mathcal{C}$ are then evaluated, as introduced in section 3.3 and only the top-scored clauses are kept for the next iteration.

[2]This corresponds to a specific *downward refinement operator* [2], which generates new specified clauses by modifying given clause.

3.3. Clause Evaluation

We now describe the clause evaluation strategy in NeSy-π and three novel metrics to evaluate clauses efficiently for predicate invention. Given the generated clauses (described in Sec. 3.2), NeSy-π scores each of them using positive and negative examples $\mathcal{D} = \{\mathcal{D}^+, \mathcal{D}^-\}$. Note that each example ($d_i^+ \in \mathcal{D}^+$ or $d_i^- \in \mathcal{D}^-$) is given as a visual scene. For clear presentation, we prepare the following notation:

- r_C is a differentiable forward reasoning function using clause C in αILP [6]. We use it to evaluate each clause. It takes visual scene as its input, converts them to probabilistic atoms, and performs differentiable forward reasoning using clause C. We consider the output as the score for clause C. The domain of r_C is $[0, 1]$.
- $r_C(d_i^+)$, $d_i^+ \in \mathcal{D}^+$ represents the confidence of clause C being true for positive example d_i^+
- $r_C(d_i^-)$, $d_i^- \in \mathcal{D}^-$ represents the confidence of clause C being true for negative example d_i^-
- $|\mathcal{D}^+|$ and $|\mathcal{D}^-|$ indicate the number of positive and negative examples in the dataset, respectively.

We introduce metric *P-Measure*, which is used to evaluate the performance of the clause on positive examples. The target clause has to be valid in all positive examples, thus the more positive examples that a clause satisfies, the better the clause is. On the contrary, the less positive examples that a clause satisfies, the worse it is.

P-Measure The *P-Measure* assesses the validation of clause on positive examples, with μ_C^P representing the P-Measure of clause C. This value indicates the cumulative confidence of positive examples that satisfy clause C with its value in the range $[0, 1]$. Since all positive examples should satisfy the target clauses, higher P-Measure of C indicates better performance.

$$\mu_C^P(\mathcal{D}^+) = \frac{1}{|\mathcal{D}^+|} \sum_{d \in \mathcal{D}^+} r_C(d) \tag{1}$$

If all the positive examples fulfill the clause C, then the P-Measure of C equals 1. This indicates that C is a *necessary condition* for the positive examples, i.e. the clause C follows the implicational relationship

$$\text{If } \mu_C^P(d) = 1, \text{ then } d \in \mathcal{D}^+ \Rightarrow r_C(d) = 1 \tag{2}$$

Similarly, we introduce metric *N-Measure*, which is used to evaluate the performance of the clause on negative examples. The target clause has to be invalid in all negative examples, thus the less negative examples that a clause does not satisfy, the better the clause is. On the contrary, the more negative examples that a clause satisfy, the worse the clause is.

N-Measure The *N-Measure* evaluates the validation of clauses on negative examples. Given a clause C, its N-Measure is denoted as μ_C^N, measuring the inverse of the cumulative confidence of negative examples that do not satisfy C. The value of μ_C^N is in the range $[0, 1]$. As the target clauses are not intended to be satisfied by all of the negative examples, a higher N-Measure of C indicates better performance. The N-Measure of C is defined as follows:

$$\mu_C^N(\mathcal{D}^-) = \frac{1}{|\mathcal{D}^-|} \sum_{d \in \mathcal{D}^-} (1 - r_C(d)) \tag{3}$$

If none of the negative examples satisfy C, the N-Measure of C is equal to 1. Assuming the examples are either positive or negative, if there is an example e that satisfies C, then d is a positive examples. The clause C is called a *sufficient condition* for the positive examples, which follows the implicational relationship

$$\text{If } \mu_C^N(d) = 1, \text{ then } r_C(d) = 1 \Rightarrow d \in \mathcal{D}^+ \tag{4}$$

If all the positive examples are satisfied by C, and no negative examples satisfy C, then C meets the definition of the target clause. In this scenario, both the P-Measure and N-Measure of C are 1, and C is a *sufficient and necessary condition* of the positive examples. This follows the implicational relationship:

$$\text{If } \mu_C^P(d) = 1 \text{ and } \mu_C^N(d) = 1, \text{ then } r_C(d) = 1 \Leftrightarrow d \in \mathcal{D}^+ \tag{5}$$

However, the majority of the clauses yield P and N measure scores between 0 and 1, i.e. they satisfy neither necessary nor sufficient condition. Since the sufficient and necessary condition is satisfied by $\mu_C^P(\mathcal{D}^+) = 1$ and $\mu_C^N(\mathcal{D}^-) = 1$, the higher the clause resulting in both P-Measure and N-Measure scores, the closer it is to the target clause. NeSy-π utilizes a best-first approach that take the both P-Measure and N-Measure into account, resulting in the following PN-Measure.

PN-Measure The *PN-Measure* evaluates a clause meeting the definition of the target clause. Given a clause C, its PN-Measure is denoted as μ_C^{PN}, measuring the product of its P-Measure and N-Measure, which is defined as follows:

$$\mu_C^{PN} = \mu_C^P \cdot \mu_C^N \tag{6}$$

To assess a set of clauses $\mathcal{C}$, NeSy-π computes μ_C^{PN} for each clause $C \in \mathcal{C}$ on the dataset, and selects the k top scoring clauses for subsequent iterations of extension or predicate invention.

3.4. Predicate Invention

NeSy-π invents new predicates by using the extended clauses $\mathcal{C}$ and their evaluations, which were described in Sec. 3.2 and Sec, 3.3, respectively. New predicates are defined by taking disjunctions of the extended clauses. We assume that all clauses in $\mathcal{C}$ have the same predicate for their head atom.

Definition 1 (Invented Predicate)**.** Given a set of clauses $\mathcal{C}$, *s.t.* $2 \leqslant |\mathcal{C}|$, any subset $\mathcal{C}_p \subseteq \mathcal{C}$, s.t. $2 \leqslant |\mathcal{C}_p| \leqslant |\mathcal{C}|$ defines a new predicate p. The clause set $\mathcal{C}_p$ is called the *explanation clause set* of predicate p. The meaning of p is interpreted as the disjunction of clauses in $\mathcal{C}_p$.

For example, assume $\mathcal{C} = \{C_1, C_2, C_3\}$ ($|\mathcal{C}| = 3$) as follows:

```
C1:   target(X) :- in(O1,X),in(O2,X),color(O1,blue),color(O2,blue).
C2:   target(X) :- in(O1,X),in(O2,X),color(O1,red),color(O2,red).
C3:   target(X) :- in(O1,X),in(O2,X),color(O1,green),color(O2,green).
```

A predicate `inv_pred_1` can be invented by disjunctively combining the first two clauses. That is, let $\mathcal{C}_p = \{C_1, C_2\}$ denote the explanation clause set of `inv_pred_1`, then `inv_pred_1` is defined as follows

```
inv_pred_1(X):- in(O1,X),in(O2,X),color(O1,blue),color(O2,blue).
inv_pred_1(X):- in(O1,X),in(O2,X),color(O1,red),color(O2,red).
```

The invented predicate `inv_pred_1` interprets the concept that *there exists a pair of objects in the image with the same color (either blue or red).* Each clause within $\mathcal{C}_p$ is referred to as a *variation*. By taking the disjunction of the clauses, the invented predicate can be assigned true if any one of its variations is true. Similarly, another predicate `inv_pred_2` can be invented by taking the disjunction of the last two clauses, `inv_pred_3` can be invented by taking the disjunction of all three clauses, and so on. In total, there are $2^{|\mathcal{C}|} - 1 - |\mathcal{C}|$ predicates that can be invented from a clause set $\mathcal{C}$ that includes at least two clauses in its explanation clause set. Due to the exponential growth rate, the invention of predicates may be restricted by a threshold, denoted as a. This threshold is defined such that $2 \leqslant a \leqslant |\mathcal{C}|$, where $|\mathcal{C}_p| \leqslant a$. As a result, subsets that surpass the cardinality of a are not utilized for predicate invention. Applying the threshold a, enforces that the possible variations of concepts are within the limit of a.

The key of predicate invention in NeSy-π is to identify the explanation clause set, which is a subset of the clause set $\mathcal{C}$ searched by the NeSy-ILP system. Due to the absence of guidance or background knowledge, the model lacks the ability to choose the optimal explanation clause set. Nonetheless, it is feasible to invent the clauses first and then evaluate them on a dataset. First, NeSy-π generates all possible clause combinations from the clause set $\mathcal{C}$, then evaluates each clause based on the dataset and finally selects the top-k highest-scoring new predicates.

Now we move on evaluating the newly invented predicates. Invented predicates can be evaluated using the P-Measure and N-Measure metrics as introduced in section 3.3. Let p be an invented predicate, $\mathcal{C}_p$ be the corresponding explanation clause set, and V_i its i-th variation.

P-Measure on Invented Predicate The P-Measure of invented predicate candidate p is utilized to evaluate the validation of an invented predicate on positive examples. The validity of predicate p increases as more positive examples are verified to hold true on it. The invented predicate is defined as the disjunction of the corresponding explanation clause set $\mathcal{C}_p$. The maximum score of the clauses in $\mathcal{C}_p$ determines the evaluation score of p, which is calculated as follows

$$\mu^P_{\mathcal{C}_p}(\mathcal{D}^+) = \frac{1}{|\mathcal{D}^+|}\sum_{d\in\mathcal{D}^+}\max_{V_i\in\mathcal{C}_p}(r_{V_i}(d)) \tag{7}$$

N-Measure on Invented Predicate The N-Measure is utilized to access the validation of an invented predicate on negative examples. The fewer the number of the negative examples in which the invented predicate holds true, the stronger its performance. The invented predicate is defined by the disjunction of the associated explanation clause set $\mathcal{C}_p$, and its evaluation score is determined by the highest-scoring clause. The score of predicate p on example d is calculated as $\max_{V_i\in\mathcal{C}_p} r_{V_i}(d)$. To indicate that the higher the value, the better, the result is computed as $1-\max_{V_i\in\mathcal{C}_p} r_{V_i}(d)$. It is also equivalent to $\min_{V_i\in\mathcal{C}_p}(1-r_{V_i}(d))$. The N-Measure for an invented predicate is determined in the following

$$\mu^N_{\mathcal{C}_p}(\mathcal{D}^-) = \frac{1}{|\mathcal{D}^-|}\sum_{d\in\mathcal{D}^-}\min_{V_i\in\mathcal{C}_p}(1-r_{V_i}(d)) \tag{8}$$

PN-Measure on Invented Predicate The PN-Measure assesses the performance of the predicate on both positive and negative examples in a quantified, objective way. This measure is derived by calculating the product of the P-Measure and the N-Measure. The formula for the PN-Measure can be written as follows:

$$\mu^{PN}_{\mathcal{C}_p} = \mu^P_{\mathcal{C}_p}\cdot\mu^N_{\mathcal{C}_p} \tag{9}$$

In NeSy-ILP systems, target clauses are extended by searching for atoms within the given language. This search relies on the assumption that all the necessary atoms are present in the given language. In contrast, NeSy-π does not require such an assumption. If necessary atoms are absent from the language, the system can invent new predicates and generate new atoms.

For instance, let us consider scenes with only two colors *red* and *blue* and two shapes *sphere* and *cube*. The pattern `three_same` represents the concept that *three objects have same color and shape simultaneously*. A possible solution for NeSy-ILP system necessitates the 3-arity predicate `two_same/2`, which is presented as the background knowledge. The target clause can be then be obtained by extending two atoms from the initial clause

```
target(X):-in(O1,X),in(O2,X),in(O3,X),two_same(O1,O2),two_same(O2,O3).
```

The predicate `two_same` is explained by five clauses, which are presented as background knowledge.

Listing 1: Background knowledge of concept `two_same`.

```
two_same(A,B):-same_color_pair(A,B),same_shape_pair(A,B),in(A,X),in(B,X).
same_shape_pair(A,B):-shape(A,sphere),shape(B,sphere),in(A,X),in(B,X).
same_shape_pair(A,B):-shape(A,cube),shape(B,cube),in(A,X),in(B,X).
same_color_pair(A,B):-color(A,red),color(B,red),in(A,X),in(B,X).
same_color_pair(A,B):-color(A,blue),color(B,blue),in(A,X),in(B,X).
```

In NeSy-π, such kind of background knowledge is not provided, but learned by the system. A possible set of rules learned by NeSy-π can be described as follows

Listing 2: Predicates invented by NeSy-π that relate to concept `two_same`

```
inv_pred1(O1,O2):-color(O1,blue),color(O2,blue),in(O1,X),in(O2,X).
inv_pred1(O1,O2):-color(O1,red),color(O2,red),in(O1,X),in(O2,X).
inv_pred2(O1,O2):-in(O1,X),in(O2,X),inv_pred1(O1,O2),shape(O1,sphere),shape(O2,sphere)
inv_pred2(O1,O2):-in(O1,X),in(O2,X),inv_pred1(O1,O2),shape(O1,cube),shape(O2,cube).
```

By updating the invented predicates for the language, NeSy-π can search for the target clause as follows:

```
target(X):-in(O1,X),in(O2,X),in(O3,X),inv_pred2(O1,O2),inv_pred2(O2,O3).
```

The predicate `target(X)` corresponds to the concept `three_same`. It takes five clauses to explain, which are learned instead of given. The predicates `inv_pred1/2` and `inv_pred2/2` are invented predicates. The other three predicates `shape/2`, `color/2`, and `in/2` are given beforehand. Predicate invention enhances the system's ability to learn new concepts with less reliance on background knowledge.

3.5. Object Grouping

In a scene with numerous objects, it is often unnecessary to consider every relationship. For instance, if there are three objects on a table, it is reasonable to query whether any two of them have the same color or same shape. However, if there are 20 objects on the table, it is not necessary to investigate whether any two of them share a property. It would be more expedient to initially consider conditions such as whether more than half or all of the objects share a property. In order to gain a comprehensive understanding of an image, it is advisable to begin by accessing it at a global level, rather than delving into its particulars right away.

For complex patterns that comprise a multitude of objects, NeSy-π utilizes an object Grouping Module (GM) that can recognise line patterns in a more efficient and organised manner. The grouping module employs the position characteristics of each object to group them accordingly. It is essential to view multiple objects as a single entity in complex scenes since the number of object relationships increases at a factorial rate, specifically $\mathcal{O}(n!)$, with n being the number of objects in a scene. Examining relationships between different groups instead of individual objects can significantly decrease the number of relations under consideration. However, not all patterns can be fully described by line groups; rather, isolated objects must be treated individually.

Five steps are required to convert an image example to group tensors $\mathcal{G}$. The main procedures of the grouping module are illustrated in Fig. 3. Additionally, the algorithm provides certain thresholds. ϵ denotes the distance threshold between an object and a group, whereas N_G is the minimum number of objects required in each group.

Object Perception. In the first step, the grouping module detects the object labels $O_{Label} \in \mathbb{Z}^{N\times 1}$ and the bounding boxes $O_{Box} \in \mathbb{R}^{N\times 4}$ in the RGB image $I_{RGB} \in \mathbb{R}^{W\times H\times 3}$ based on an object detector, such as Mask-RCNN [10], where N is the number of objects detected, W and H are the width and height of the image. Based on the camera matrix K, the 3D coordinates of all the pixels $I_{3D} \in \mathbb{R}^{W\times H\times 3}$ relative to the camera coordinate system can be calculated from the depth map $I_{Depth} \in \mathbb{R}^{W\times H\times 1}$. Since in RGB-D images, the pixels on I_{RGB} and I_{Depth} correspond to each other, each bounding box in O_{Box} corresponds to the same region in I_{3D} based on the pixel coordinate system.

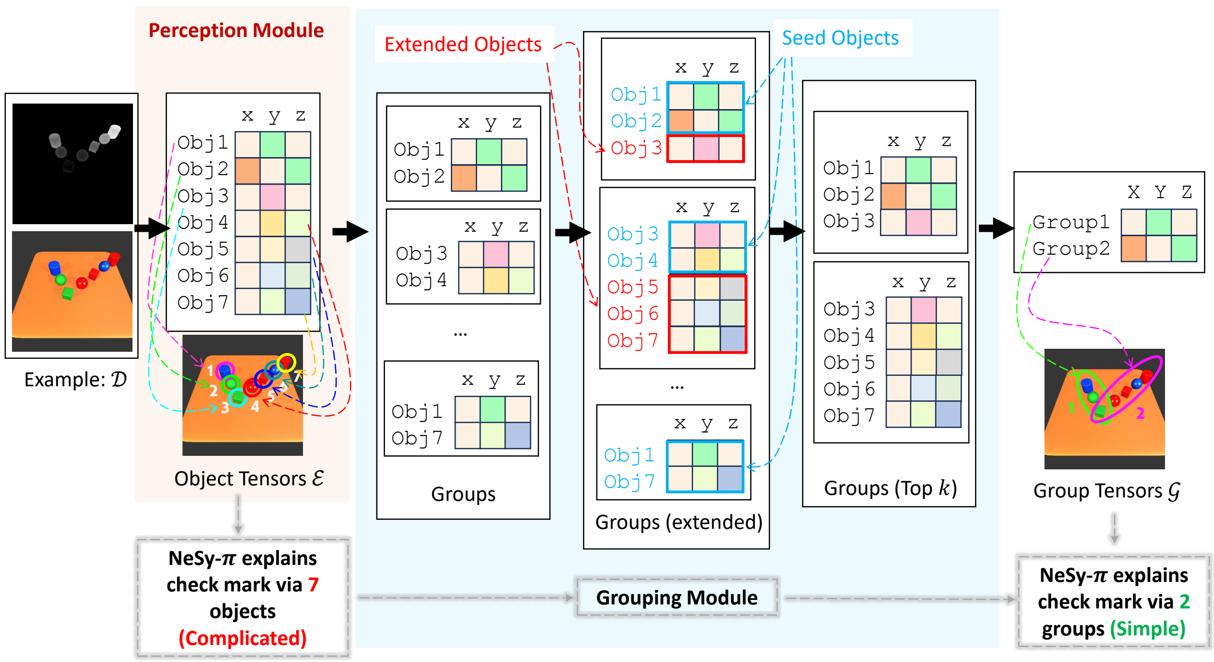

Perception Module
Example: $\mathcal{D}$
Object Tensors $\mathcal{E}$
Extended Objects
Seed Objects
Groups
Groups (extended)
Groups (Top k)
Group Tensors $\mathcal{G}$
Grouping Module
NeSy-π explains check mark via 7 objects (Complicated)
NeSy-π explains check mark via 2 groups (Simple)

Fig. 3. **Grouping Module**. The grouping module takes an image as input and produces the group tensors $\mathcal{G}$ in 5 stages. (1) The perception module detects objects and encodes them as tensors $\mathcal{E}$; (2) Objects are chosen based on their properties to generate small groups. (3) Extend each small group with as many objects as possible. (4) Evaluate and select the highest-scoring groups. (5) Encode groups into group tensors $\mathcal{G}$.

This allows the object surface positions to be recorded relative to the camera coordinate system. For each object there is a set of surface points. The object position can be approximated by the median number of these points. For all the objects, their positions are given by $O_{3D} \in \mathbb{R}^{N\times 3}$. Given O_{3D} and O_{Label}, objects can be encoded into tensors $\mathcal{E} \in \mathbb{R}^{N\times M}$, where M is the number of properties for each object. In NeSy-π, the properties taken into account are *position, color, and class*. The position is specified by the 3D coordinates of the object, while both colors and classes are one-hot coded.

Group Generation NeSy-π proposes an approach to gather objects into line groups based on their 3D positions. The strategy is to first generate small groups and then expand them to the maximum size. Since two points define a line, a small group is generated by two objects. The object positions can also be used to determine the corresponding line function. For N objects, $\binom{N}{2}$ small line groups are generated. As shown in Fig. 3, seven objects are placed on the table in the given example, thus $\binom{7}{2} = 26$ small groups are generated from this scene.

Group Extension The groups are further extended based on their line functions. For each small group, the remaining $N-2$ objects not contained in the groups can be evaluated based on their distances to the line. By specifying a threshold parameter ϵ, the objects with have distances within ϵ can be extended to the groups. As shown in Fig. 3, `Obj1` and `Obj2` can generate a small group. Let l_{12} be the line defined by this group, the system will calculate the distance between the remaining objects `Obj3`, `Obj4`, `Obj5`, `Obj6`, `Obj7` to l_{12}, and then compare the distances with the threshold ϵ. By setting a suitable threshold, `Obj3` is extended to the group. Similarly, the small group created by `Obj3` and `Obj4` can be extended with `Obj5`, `Obj6` and `Obj7`.

Group Evaluation Since the motivation of grouping module is to decrease the relations between objects, it is important to minimize the number of groups. After the group generation, $\binom{N}{2}$ group are generated and then expanded to their maximum potential. The group evaluation removes duplicate groups. Additionally, a minimum group size of N_G is established, which serves as a system parameter to remove groups with fewer objects than the threshold. Choosing a smaller value of N_G enables the explanation of rules in great detail, but at the cost of increased time and space requirements. Conversely, choosing a larger value of N_G leads to faster reasoning, but pattern details may be overlooked. For our experiment, we opted for the smallest possible value of N_G that still maintained acceptable reasoning times in order to enhance its robustness.

Group Encoding Finally, the groups are encoded as tensors $\mathcal{G}$. The position of each group is determined by averaging the positions of objects within the group.

The group module is an optional module of NeSy-π. It is used for scenes with potential group structures, like lines in this given example. By generating rules at group level, the logical structures between objects can be expressed much more clearly and easily. The invented predicates are also invented at the group level.

3.6. NeSy-π Algorithm

Algorithm 1 provides the pseudo code of the process of predicate invention utilizing the NeSy-ILP and NeSy-π systems. The NeSy-π system focuses on extending the initial language $\mathcal{L}_0$ to the output language $\mathcal{L}_{out}$ through the invention of new predicates. Meanwhile, NeSy-ILP is tasked with searching for target clauses.

The algorithm includes a grouping module as discussed in Sec. 3.2-3.5. The initial language $\mathcal{L}_0$ includes all the background predicates alongside the initial clause $\mathcal{C}_0$, namely `target(X) :-in(O1,X)`.. Additionally, the camera matrix K, image dataset $\mathcal{I}_{RGB}$ and the corresponding depth map $\mathcal{I}_{Depth}$ are also inputs. The algorithm's output is the extended language $\mathcal{L}_{out}$, which includes all invented and background predicates.

(Line 1–4) Produce object tensors $\mathcal{E}$ from the input dataset $\mathcal{I}_{RGB}$ and $\mathcal{I}_{Depth}$. An object detector, like Mask R-CNN[10], is employed to identify object labels $\mathcal{O}_{Label}$ and bounding boxes $\mathcal{O}_{Box}$ in the RGB images $\mathcal{I}_{RGB}$. The 3D coordinates of all pixels $\mathcal{I}_{3D}$ can then be transformed, relative to the camera's coordinate system, from the depth map $\mathcal{I}_{Depth}$ and the camera matrix K. The 3D position of the objects $\mathcal{O}_{3D}$ can be determined by taking the median value of the corresponding region in $\mathcal{I}_{3D}$ with respect to corresponding bounding box $\mathcal{O}_{Box}$. **(Line 5–7)** Group the objects based on their positions $\mathcal{O}_{3D}$. **(Line 9–12)** The iteration variable N is responsible for specifying the number of groups covered by the searched clauses. The given parameter N_{max_groups} controls the maximum number of groups covered by the clauses, which is set with respect to the complexity of the training patterns. For complex Kandinsky Patterns, this parameter is set to 2, while for Alphabet Patterns, it is set to 5 or 6, as detailed in section 4.1. Additionally, two lists are initialized. $\mathcal{P}_{inv\_all}$ serves as a repository for invented predicates, while $\mathcal{P}_{bk}$ is the set of all background predicates in the initial language $\mathcal{L}_0$. **(Line 14–17)** In each iteration, a background predicate P is selected from $\mathcal{P}_{bk}$. A new language $\mathcal{L}$ is instantiated with no pre-existing predicate. P is the first predicate that is used to update this language. **(Line 19)** The for-loop spanning lines 19-38 comprises the clause extension (line 22-29) and predicate invention (line 33-37). N_{step_max} determines the maximum length of the clauses that can be extended. **(Line 22-31)** Extend the clauses in the clause set $\mathcal{C}$ for N_{step_max} iterations. During the first iteration, the clause in $\mathcal{C}$ serves as the initial clause. The clauses are subsequently extended with new atoms, followed by the evaluation of the whole dataset. The top-scoring clauses are retained while the rest is pruned. If a target clause has been identified, then the for loop is terminated. **(Line 32-37)** After the clause extension, invent the predicates. Each element in $\mathbf{C}$ is a subset of extended clauses $\mathcal{C}$, as well as an explanation clause set. The new predicates $\mathcal{P}_{inv}$ are generated from $\mathbf{C}$. The evaluation of the invented predicates is based on the clauses in the corresponding explanation clause set. The score of these clauses can be obtained directly from μ_{PN}. By limiting the number of invented predicates, only the top-k invented predicates are retained. The invented predicates are then added to the list $\mathcal{P}_{inv\_all}$ and the language $\mathcal{L}$. **(Line 44)** The algorithm returns a language $\mathcal{L}_{out}$ consisting of all the remaining invented predicates from $\mathcal{P}_{inv\_all}$.

4. Experimental Evaluation

NeSy-π aims to aid NeSy-ILP systems such as αILP [6] in tackling visual reasoning tasks. Frequently used evaluation datasets are the 2D dataset Kandinsky Patterns [9], and the 3D dataset CLEVR-Hans [11], which is a variant of CLEVR [7]. Both datasets utilize geometry objects to represent fundamental concepts of common sense. In this paper, we propose three datasets for evaluating NeSy-π, which follow the similar dataset configuration with αILP [6]. The calibrated RGB-D camera captures all the scenes. The RGB images serve as input for the perception

Algorithm 1 Predicate Invention on Visual Scenes using NeSy-ILP and NeSy-π

```
Require: $\mathcal{L}_0$, $\mathcal{C}_0$, $\mathcal{I}_{RGB}$, $\mathcal{I}_{Depth}$, $K$, $N_{max\_groups}$, $N_{max\_c\_length}$

1: // Perception Model takes RGB-D images as input to detect the objects and their 3D positions, the predicted
   results are saved as tensors.
2: $\mathcal{O}_{Label}, \mathcal{O}_{Box} \leftarrow$ object_detector($\mathcal{I}_{RGB}$)          /* Perception Model */
3: $\mathcal{O}_{3D} \leftarrow$ depth2point_cloud($\mathcal{I}_{Depth}, \mathcal{O}_{Box}, K$)
4: $\mathcal{E} \leftarrow \{\mathcal{O}_{3D}, \mathcal{O}_{Label}\}$
5: // Grouping model gather objects into groups.
6: $\mathcal{G} \leftarrow$ group_objs($\mathcal{E}$)          /* Grouping Module */
7: $\mathcal{G}_{top} \leftarrow$ top_groups($\mathcal{G}$)
8: // $N$ controls the number of groups covered by the clauses.
9: for $1 \leqslant N < N_{max\_groups}$ do
10:     // $\mathcal{P}_{inv\_all}$ saves all of the invented predicates. $\mathcal{P}_{bk}$ saves all the background predicates in the given language.
11:     $\mathcal{P}_{inv\_all} \leftarrow \phi$
12:     $\mathcal{P}_{bk} \leftarrow$ extract_bk_predicates($\mathcal{L}_0$)
13:     // Iterate over each predicate $P$ in $\mathcal{P}_{bk}$.
14:     for P $\in \mathcal{P}_{bk}$ do          /* Target Clause Searching */
15:         $\mathcal{L} \leftarrow$ new_language()
16:         $\mathcal{L} \leftarrow$ update($\mathcal{L}, P$)
17:         $\mathcal{C} \leftarrow \mathcal{C}_0$
18:         // The maximum clause length is controlled by $N_{step\_max}$.
19:         for $1 \leqslant N_{step\_max} \leqslant N_{max\_c\_length}$ do
20:             // Extend the clauses with predicate $P$ for $N_{step\_max}$ times.
21:             // In each extension, new clauses are evaluated. Check if any of them are target clauses.
22:             for $1 \leqslant N_{step} \leqslant N_{step\_max}$ do          /* Extend and Evaluate Clauses */
23:                 $\mathcal{C} \leftarrow$ extend($\mathcal{C}, \mathcal{L}$)
24:                 $\mu_{PN} \leftarrow$ eval_clauses($\mathcal{C}, \mathcal{G}_{top}$)
25:                 $\mathcal{C}, \mu_{PN} \leftarrow$ prune($\mathcal{C}, \mu_{PN}$)
26:                 $done \leftarrow$ check_result($\mu_{PN}$)
27:                 if $done$ then break
28:                 end if
29:             end for
30:             if $done$ then break
31:             end if
32:             // Invent new predicates based on clause set $\mathbf{C}$. The new predicates are added to the language $\mathcal{L}_P$.
33:             $\mathbf{C} \leftarrow$ combination($\mathcal{C}$)          /* Invent New Predicates */
34:             $\mathcal{P}_{inv} \leftarrow$ generate_predicate($\mathbf{C}$)
35:             $\mathcal{P}_{inv} \leftarrow$ top_k($\mathcal{P}_{inv}, \mu_{PN}$)
36:             $\mathcal{P}_{inv\_all} \leftarrow \mathcal{P}_{inv} + \mathcal{P}_{inv\_all}$
37:             $\mathcal{L} \leftarrow$ update($\mathcal{L}, \mathcal{P}_{inv}$)
38:         end for
39:         if $done$ then break
40:         end if
41:     end for
42: end for
43: // Update all invented predicates to language $\mathcal{L}_{out}$.
44: $\mathcal{L}_{out} \leftarrow$ update($\mathcal{L}_0, \mathcal{P}_{inv\_all}$)
45: return $\mathcal{L}_{out}$
```

module. The system locates the 3D positions of the objects relative to the camera coordinate system using the corresponding depth maps and camera matrix.

4.1. Datasets

The datasets comprise synthetic images generated using the *Unity game engine*[12]. In every image, objects are positioned on a table and captured by a calibrated RGB-D camera. A *scene* in the dataset refers to an image captured by the camera. Every scene is made up of basic objects that represent logical concepts. The object categories comprise of three types: *sphere*, *cube*, and *cylinder*; the objects come in three colors: *red*, *green*, and *blue*. The size and surface texture of the objects remain consistent across all scenes. A *pattern* refers to a set of scenes consisting of both positive scenes and negative scenes. All of the positive scenes follow a common logical concept, while none of the negative scenes do. To examine the predicate invention performance of the NeSy-π, every pattern in the dataset is accompanied by at least one new concept not included in the initial language.

Simple Kandinsky Patterns This dataset represents four logical patterns, as shown in Fig. 4. The patterns comprises a maximum of four objects. In each pattern, several objects are placed on the table. The details of each pattern is introduced as follows:

(*Close*) In positive scenes, two objects are located close to each other. The maximum distance between two objects is one-seventh of the table's width. Negative scenes are two objects that are far apart, with a minimum distance between them of half of the table's width.

(*Diagonal*) Positive scenes require two objects to be positioned diagonally from each other, while negative scenes involve two objects being positioned horizontally.

(*Three Same*) Three objects appear in scenes, positive scenes require that all objects possess the same color and shape. Negative scenes, however, involve three objects that have at least two differences in color or shape.

(*Two Pairs*) In scenes with two pairs of objects, positive scenes require that each pair share the same color and shape. Negative scenes require at most one pair that shares the same color and shape.

Complex Kandinsky Patterns This dataset uses objects to represent four logical patterns, as shown in Fig. 5. Compared to the simple Kandinsky patterns, the complex Kandinsky patterns consists of a considerately larger number of objects and relations among objects. The objects are ordered according to certain spatial rules, such as *perpendicular*, *parallel* and so on. The details of each pattern is introduced as follows:

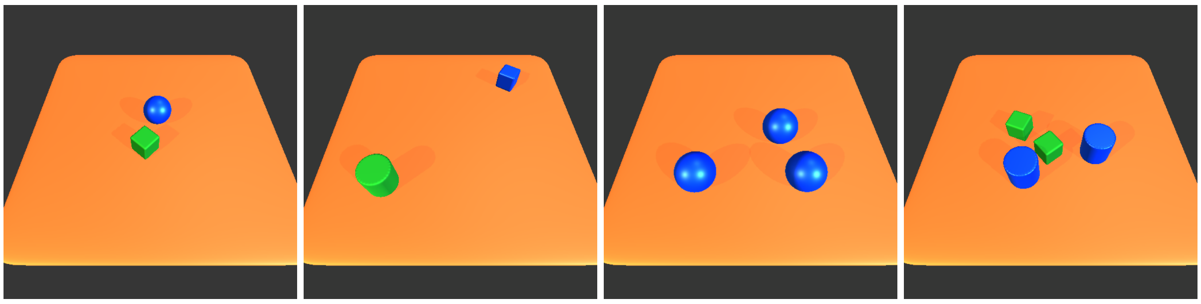
(*Check Mark*) In positive scenes, five to seven objects form a check mark shape. The direction of the check mark is rotated in range 0 to 45 degrees in each scene. In negative scenes, five to eight objects form two intersecting lines, with the intersection point never being an endpoint of both lines simultaneously.

(*Parallel*) Positive scenes consists of six to eight objects arranged in two parallel rows. Negative scenes consist of six to eight objects arranged in two rows with an angle between them ranging from 20 to 80 degrees.

(*Perpendicular*) In positive scenes, six to eight objects are arranged in two perpendicular rows. Negative scenes consist of six to eight objects arranged in two rows with angle between them ranging from 20 to 80 degrees.

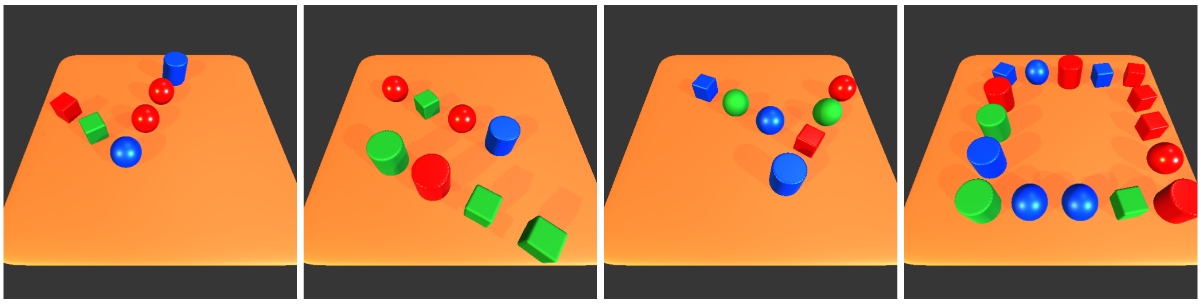
(*Square*) In positive scenes, several objects form the shape of a square, with consistent spacing between adjacent objects on each edge. The square size in each scene range from 3, 4, or 5 objects per edge. In negative scenes, the objects are arranged in the shape of a rectangle, with a consistent spacing between adjacent objects on each edge. The longer side of the rectangle consists of 5 objects, while the shorter side consists of 3 objects. The rectangle is located either horizontally or vertically.

Latin Alphabet Patterns This dataset uses objects representing 26 Latin letters. Fig. 15 shows some examples of the letter patterns. A complete version is shown in the Appendix B. These patterns are composed of line-based elements. In each pattern, a set of objects is placed on a table and forms the shape of a letter in the Latin alphabet. In order to introduce more variations for each pattern, letter cases (upper case and lower case) are considered, i.e. each pattern has two variations. The colors and shapes of the objects are changed randomly in each scene. The center position of all objects is also shifted slightly in each scene. The dataset of each pattern consists of the same number of positive and negative scenes. The positive scenes are created based on the two variations of that pattern, and the negative scenes are created based on the variations of all the other patterns.



(a) Close (b) Diagonal (c) Three Same (d) Two Pairs

Fig. 4. **Simple Kandinsky Patterns**. These patterns are designed with only a few objects. Each pattern refers to a logical concept, as indicated by the name, which is not explained by background knowledge. **Close**: Two objects are close to each other. **Diagonal**: Two objects are diagonally located. **Same**: Three objects have the same shape and the same color. **Two Pairs**: Two pairs of objects are in the scene. A pair is defined by two objects with the same color and shape.



(a) Check Mark (b) Parallel (c) Perpendicular (d) Square

Fig. 5. **Complex Kandinsky Patterns**. These patterns are designed with at least five objects in the scenes. These objects form 2D shapes by their position. **Check Mark**: Several objects form the shape of check mark. **Parallel**: Multiple objects are arranged in two parallel rows. **Perpendicular**: Multiple objects are arranged in two perpendicular rows. **Square**: Several objects are arranged in the shape of a square.

4.2. Evaluation Results

The experiments aim to answer the following questions:

Q1. Can NeSy-π invent useful predicates from visual scenes improving the performance of NeSy-ILP systems?
Q2. Can NeSy-π solve complex patterns that cannot be solved by the state-of-the-art NeSy-ILP baseline?
Q3. Is NeSy-π computationally efficient?
Q4. Is NeSy-π robust to unseen falsifying examples?

A1 (NeSy-π Effectiveness): The background predicates utilized for solving the patterns in this work are listed in Table 1. All predicates invented by NeSy-π are based on these predicates. Moreover, no other predicates or background clauses are provided for the reasoning process. In αILP [6], certain concepts such as `same_color`, `same_shape` are explained using a set of background clauses, but no such clauses are provided for reasoning in NeSy-π. However, certain background predicates necessitate parameters determination for precision purposes. The parameter N_ρ decides the precision of the distance predicate (`rho/3`). For example, $N_\rho = 20$ means that the distance between two objects can be characterized from `rho0` ($[0.00W, 0.05W]$) to `rho19` ($[0.95W, 1.00W]$) with W as the table width. Similarly, the precision of the direction predicate (`phi/3`) is determined by the parameter N_ϕ, while the precision of the slope predicate (`slope/2`) is determined by the parameter N_{slope}. These parameters determine the granularity of the concepts that can be handled in the system.

Fig. 7 shows an example of the learning result for the simple Kandinsky pattern *close*. The target clause is searched by αILP, and the used predicate `inv_pred253` is invented by NeSy-π, which can be interpreted by *the distance*

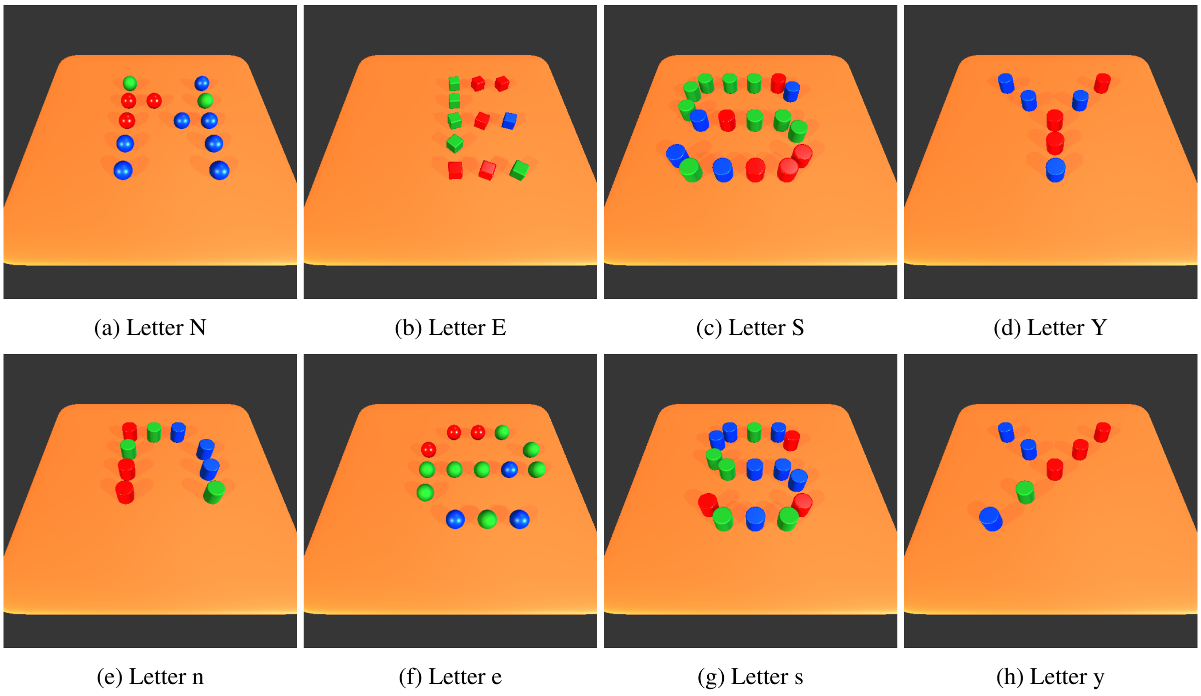


Fig. 6. **Latin Alphabet Patterns.** Latin letters (including upper and lower case) consisting of multiple objects. The shape of the individual letters consists mainly of lines. For more examples, see appendix B.

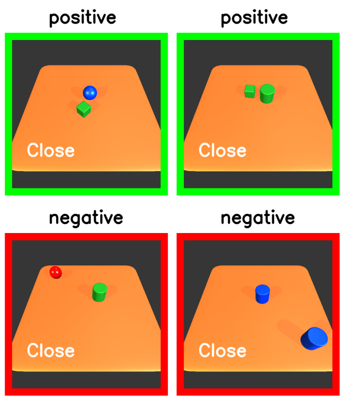
Predicate	Description
`in/2/O,X`	Evaluate whether the object/group `O` exists in the image `X`.
`shape/2/O,S`	Evaluate whether the object/group `O` is of the form `S`.
`color/2/O,C`	Evaluate if the object `O` has color `C`.
$\texttt{phi/3/O}_1\texttt{,O}_2\texttt{,}\phi$	Evaluate whether the direction of object/group $\texttt{O}_2$ corresponds to the object/group $\texttt{O}_1$ falls within the range of ϕ.
$\texttt{rho/3/O}_1\texttt{,O}_2\texttt{,}\rho$	Evaluate whether the distance between the object $\texttt{O}_2$ and $\texttt{O}_1$ falls within the range of ρ.
`slope/2/G,k`	Evaluate whether the slope of object group `G` falls within the range of `k`.

Table 1
Background predicates for solving logical patterns.

between object O1 and O2 is in the range `rho4` *to* `rho8`. The invented predicates that are not used in the target clauses are not listed in the figure. Appendix B shows the corresponding target clauses and the invented clauses of the rest patterns.

Table 2 shows the evaluation result of all simple Kandinsky patterns based on the proposed metrics in section 3.3. NeSy-π has successfully invented new predicates and found the promising target clauses for all simple patterns, resulting in PN-Measure values over 100% higher than the baseline model αILP, with the exception of the *diagonal* pattern at only 14.94%. This is due to the limited variations in the *diagonal* pattern compared to others. Only two variations exist in the positive scenes (since a square has two diagonals), and two variations exist in the negative scenes (forming a row either at the top or bottom of the table). Other patterns, such as *close*, have randomly changed object distances within a specified range. Consequently, there are many more variations in these patterns. PN-Measure is a balanced measurement that considers both positive and negative examples simultaneously. Therefore, the outperforming of PN-Measure does not guarantee the outperforming of both P-Measure and N-Measure.

A2 (Complex Patterns): For the complex Kandinsky patterns, NeSy-π uses the *Grouping Module* to gather the objects into groups. The predicates are invented on the group level. In the experiment, the maximum group number of the grouping module, i.e. the parameter N in algorithm 1, is given as 5. The dataset *Complex Kandinsky Patterns*



```
(Close)

target(X):-in(O1,X),in(O2,X),inv_pred253(O1,O2).
inv_pred253(O1,O2):-in(O1,X),in(O2,X),rho(O1,O2,rho4).
inv_pred253(O1,O2):-in(O1,X),in(O2,X),rho(O1,O2,rho5).
inv_pred253(O1,O2):-in(O1,X),in(O2,X),rho(O1,O2,rho6).
inv_pred253(O1,O2):-in(O1,X),in(O2,X),rho(O1,O2,rho7).
inv_pred253(O1,O2):-in(O1,X),in(O2,X),rho(O1,O2,rho8).

N_rho = 20
```

Fig. 7. **Pattern Close.** The logic of the pattern is that two objects are close to each other. **Left:** Two positive (green border) and two negative (red border) examples of the pattern *Close*. **Right**: Learned target clauses and the corresponding invented predicates of the pattern *Close*. The distance precision parameter N_ρ is given as 20. The predicate `inv_pred253` evaluates if the distance of `O1` and `O2` in range $rho4 \sim rho8$ (15% $\sim$ 40% of table width).

Pattern	#Object	#Image	P-Measure			N-Measure			PN-Measure			#Inv_Pred
			α	π	Δ	α	π	Δ	α	π	Δ	
Close	2	100	0.43	1.00	132.56%	1.00	0.91	-9.00%	0.43	0.91	111.63%	1
Diagonal	2	100	0.87	1.00	14.94%	1.00	1.00	0.00%	0.87	1.00	14.94%	1
Same	3	100	0.50	0.92	84.00%	0.67	1.00	49.25%	0.34	0.92	170.59%	2
Two Pairs	4	100	0.64	0.96	50.00%	0.68	1.00	47.06%	0.44	0.96	118.18%	2

Table 2

Learning result on Simple Kandinsky Patterns. α is the baseline approach based on αILP [6], π is the proposed approach using αILP with NeSy-π, Δ is the improvement in percentage calculated as $(\pi - \alpha)/\alpha \times 100\%$, #Inv_Pred is the number of invented predicates used in the target clauses.

Pattern	#Object	#Image	P-Measure			N-Measure			PN-Measure			#Inv_Pred
			α	π	Δ	α	π	Δ	α	π	Δ	
Check Mark	5-6	100	0.66	0.93	40.91%	0.90	0.99	10.00%	0.59	0.92	55.93%	2
Parallel	6-8	100	0.33	0.93	181.82%	1.00	1.00	0.00%	0.33	0.93	181.82%	1
Perpendicular	6-8	100	0.63	0.92	46.03%	0.88	1.00	13.64%	0.55	0.92	67.27%	2
Square	8-16	100	0.50	0.83	66.00%	0.72	1.00	38.89%	0.36	0.83	130.56%	1

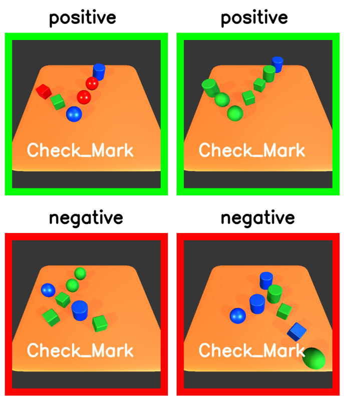
Table 3

Learning result on Complex Kandinsky Patterns. α is the baseline approach based on αILP [6], π is the proposed approach using αILP with NeSy-π, Δ is the improvement in percentage calculated as $(\pi - \alpha)/\alpha \times 100\%$, #Inv_Pred is the number of invented predicates used in the target clauses.

uses objects representing basic line relations and other basic geometry shapes. The given background knowledge is the given predicates shown in Table 1, and the line grouping module explained in section 3.5.

Fig. 8 lists the learned target clauses and the corresponding invented predicates of the pattern *check mark*. The target clause is described by 2 invented predicates. The used invented predicate `inv_pred0` is explained as follows: *the group O2 is either in the direction* `phi3`, `phi4`, `phi10`, *or* `phi12` *of O1*, which corresponds to the direction ranges $[36°, 54°]$, $[54°, 72°]$, $[162°, 180°]$, and $[198°, 216°]$ respectively. The predicate `inv_pred211` is explained as follows: *the distance between two groups is in the range* `rho4` *to* `rho7`, which corresponds to the distance ranges $[0.15W, 0.35W]$ with W as the table width. Appendix B shows the learned clauses of the remaining three complex Kandinsky patterns. Table 3 shows the scores of each pattern based on proposed metrics. NeSy-π improves the scores in both P-Measure and N-Measure with using 1-2 invented predicates in the target clauses.

Furthermore, we have also analyzed the more intricate dataset *Alphabet Patterns*, which consists of 5 to 20



```
(Check Mark)

target(X):-in(O1,X),in(O2,X),inv_pred0(O1,O2),
inv_pred211(O1,O2).
inv_pred0(O1,O2):-in(O1,X),in(O2,X),phi(O1,O2,phi3).
inv_pred0(O1,O2):-in(O1,X),in(O2,X),phi(O1,O2,phi10).
inv_pred0(O1,O2):-in(O1,X),in(O2,X),phi(O1,O2,phi4).
inv_pred0(O1,O2):-in(O1,X),in(O2,X),phi(O1,O2,phi12).
inv_pred211(O1,O2):-in(O1,X),in(O2,X),rho(O1,O2,rho5).
inv_pred211(O1,O2):-in(O1,X),in(O2,X),rho(O1,O2,rho4).
inv_pred211(O1,O2):-in(O1,X),in(O2,X),rho(O1,O2,rho6).
inv_pred211(O1,O2):-in(O1,X),in(O2,X),rho(O1,O2,rho7).

N_rho = 20, N_phi = 20, N_slope = 8
```

Fig. 8. **Pattern Check Mark.** The logic of the pattern is that several objects form a shape of check mark. The number of objects on the right side of check mark changes randomly from 3 to 5 in each scene. **Left:** Two positive (green border) and two negative (red border) examples of pattern *Check Mark*. **Right**: Learned target clauses and the corresponding invented predicates of the pattern *Check Mark*. The predicate `inv_pred0` evaluates if the direction of `O1` and `O2` in range $phi3/phi4/phi10/phi12$.

objects. The learning result of certain letters on NeSy-π with αILP systems are shown in appendix B. Table 4 illustrates the best learning result of each patterns on two systems. NeSy-π shows an average PN-Measure score of 0.911, exhibiting a 6.55% improvement compared to αILP. The average P-Measure is 0.942 and N-Measure is 0.987. Certain letters achieves even 20% to 30% improvement on PN-Measure, such as letter I, L, M and R. Some letters achieves only minor or even negative improvement, that is because the baseline model αILP already achieves high scores (mostly higher than 0.9), thus the improvement spaces left to the NeSy-π are limited, such as letter D, H, K. In addition, NeSy-π does not guarantee a higher PN-Measure comparing to αILP alone, such as letter C. That is because the clauses are extended in a best-first approach in each step. The predicates in two systems are different, thus the best atoms for extension in two systems can be different. It is possible that a clause with invented predicates achieves high score in the previous steps, but increases only minor score in the following steps. In this case, the target clauses still contain the invented predicate, but its measurement can be smaller than the clause without invented predicate. In total, there is 1 pattern that does not invent any predicate, 11 patterns invent one predicate, 9 patterns invent two predicates and 5 patterns invent three predicates. On average, 1.65 new predicates are invented to describe each pattern.

A3 (NeSy-π Efficiency): NeSy-π can improve the time efficiency for αILP system. The invented predicates improve the expressiveness of the language so that the rules can be expressed by shorter clauses. Consequently, the algorithm needs fewer iterations to search the target clauses, thereby improving the time efficiency.

In Fig. 9, the graph displays the runtime for each alphabet pattern. The height of each bar represents the corresponding letter's run-time, with the color indicating the number of groups N covered by the respective target clause. The fewer groups the target clause covers, the fewer iterations the algorithm requires. Fig. 9 (c) and (d) show the ranking of the training time in descending order. On average, the NeSy-π+αILP method requires 64 minutes and 47 seconds to solve one letter, while the αILP method takes 126 minutes and 28 seconds to solve one letter. Therefore, NeSy-π is roughly twice as fast. On the other hand, it is apparent that fewer iteration N adheres to the less training time. The letter B has the highest iteration and also the longest training time. On the other hand, letters such as U, X have the fewest iterations and, as a result, have the fastest training time. On average, αILP with NeSy-π module requires 0.46 fewer iterations than αILP alone.

A4 (NeSy-π Robustness): In this experiment, we show that NeSy-π is robust to falsifying patterns that have not seen in the training phase, *i.e.* NeSy-π exhibits robustness in distinguishing similar positive and negative examples in the dataset, achieving high accuracy despite their similarity.

Pattern	#Object	#Image	P-Measure			N-Measure			PN-Measure			#Inv_Pred
			α	π	Δ	α	π	Δ	α	π	Δ	
Letter A	14	100	0.92	0.86	-6.52%	0.85	0.96	12.94%	0.78	0.83	5.58%	1
Letter B	13-20	100	0.93	0.93	0.00%	1.00	1.00	0.00%	0.93	0.93	0.00%	0
Letter C	9-12	100	0.96	0.96	0.00%	1.00	0.99	-1.00%	0.96	0.95	-1.00%	3
Letter D	13-14	100	0.95	0.95	0.00%	0.98	1.00	2.04%	0.93	0.95	2.04%	2
Letter E	11-14	100	0.96	0.96	0.00%	1.00	1.00	0.00%	0.96	0.96	0.00%	1
Letter F	9-10	100	0.96	0.88	-8.33%	0.79	0.95	20.25%	0.76	0.84	10.23 %	1
Letter G	16-18	100	0.96	0.96	0.00%	0.86	0.98	13.95%	0.83	0.94	13.95%	1
Letter H	11-12	100	0.96	0.91	-5.21%	0.93	1.00	7.53%	0.89	0.91	1.93%	1
Letter I	9	100	0.76	0.93	22.37%	0.94	0.94	0.00%	0.71	0.87	22.37%	1
Letter J	11	100	0.96	0.93	-3.13%	0.91	0.99	8.79%	0.87	0.92	5.39%	2
Letter K	11-14	100	0.90	0.94	4.44%	1.00	1.00	0.00%	0.90	0.94	4.44%	2
Letter L	7-8	100	0.71	0.96	35.21%	0.96	0.94	-2.08%	0.68	0.90	32.39%	2
Letter M	11-13	100	0.78	0.90	15.38%	0.86	0.97	12.79%	0.67	0.87	30.14%	1
Letter N	9-12	100	0.96	0.96	0.00 %	0.98	0.98	0.00 %	0.94	0.94	0.00%	3
Letter O	12-16	100	0.94	0.95	1.06%	1.00	1.00	0.00%	0.94	0.95	1.06%	1
Letter P	14-15	100	0.93	0.94	1.08%	1.00	1.00	0.00%	0.93	0.94	1.08%	3
Letter Q	15	100	0.98	0.98	0.00%	1.00	1.00	0.00%	0.98	0.98	0.00%	2
Letter R	8-18	100	0.88	0.90	2.27%	0.79	0.99	25.32%	0.70	0.89	28.16%	3
Letter S	15-18	100	0.96	0.95	-1.04%	0.99	1.00	1.01%	0.95	0.95	0.00%	2
Letter T	9	100	0.96	0.96	0.00%	0.97	1.00	3.09%	0.93	0.96	3.09%	2
Letter U	11-13	100	0.95	0.95	0.00%	0.95	1.00	5.26%	0.90	0.95	5.26%	3
Letter V	5-9	100	0.96	0.96	0.00%	0.98	1.00	2.04%	0.94	0.96	2.04%	1
Letter W	9-13	100	0.95	0.95	0.00%	0.83	0.97	16.87%	0.79	0.92	16.87%	2
Letter X	5-9	100	0.96	0.96	0.00%	1.00	1.00	0.00%	0.96	0.96	0.00%	1
Letter Y	7	100	0.88	1.00	13.64%	1.00	1.00	0.00%	0.88	1.00	13.64%	2
Letter Z	10-13	100	0.95	0.96	1.05%	1.00	1.00	0.00%	0.95	0.96	1.05%	1
Average	11.73	100	0.922	0.942	**2.169%**	0.945	0.987	**4.444%**	0.855	0.911	**6.550%**	1.692

Table 4

Learning result of NeSy-π on Latin Alphabet Patterns. α is the base-line approach based on αILP [6], π is the proposed approach using αILP with NeSy-π, Δ is the improvement in percentage calculated as $(\pi - \alpha)/\alpha \times 100\%$, $\#$Inv_Pred is the number of invented predicates used in the target clauses.

Dataset. In order to evaluate the robustness of the learned clauses, another test dataset for the alphabet patterns is generated with similar positive and negative scenes. The positive scenes are still generated from the original letter patterns. However, the negative scenes are generated by intervening the positive scenes, *i.e.* randomly removing one object from the positive patterns, as shown in Fig. 10. Such negative patterns are not seen by the model during training. To solve this task correctly, models need to be keen on small violations of the learned logical concepts.

Result. Fig. 11 shows the evaluation result for the robustness test dataset. In comparison to the efficiency result in *A3*, when negative examples with similar patterns as positive examples are used, NeSy-π performance on N-Measure decreases from 0.98 to 0.77. Additionally, none of the test image patterns were seen by the model during the training period, and they are also very similar to the positive patterns (with only one object removed). In addition, when the robustness test dataset is applied to the clauses acquired through αILP, the N-Measure average is approximately 0.70, as demonstrated in the right plot of Fig. 11. In this case, the NeSy-π attains a higher N-Measure.

5. Related Work

We revisit relevant studies of NeSy-π.

Inductive Logic Programming and Predicate Invention. Inductive Logic Programming (ILP) [1–3, 13] has emerged at the intersection of machine learning and logic programming. ILP learns generalized logic rules given

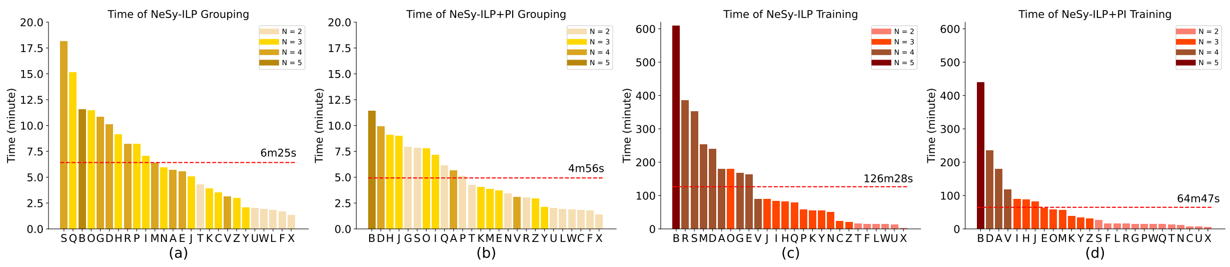


Fig. 9. **Time consumption on Alphabet Patterns.** Time consumption of the individual letters in descending order. The red dashed lines show the average time for each plot. The colors of the bars differ according to the number of groups covered by the target clauses. (a). Grouping time of αILP. (b). Grouping time of αILP with NeSy-π. (c). Training time of αILP. (d). Training time of αILP with NeSy-π.

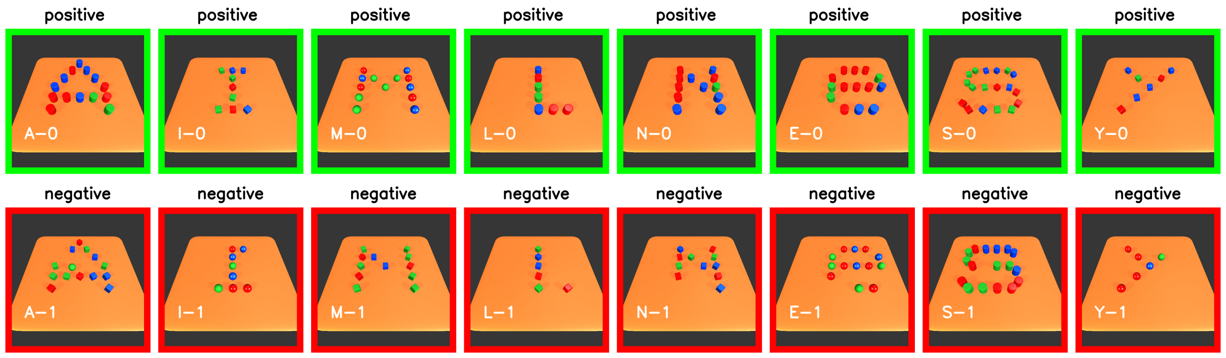


Fig. 10. **Examples of Robustness Test Dataset.** The top row displays a selection of the original letter patterns. The bottom row shows negative letter patterns with one object randomly removed. To solve this dataset correctly, the model needs to be robust to the falsifying negative scenes not appeared in the training phase.

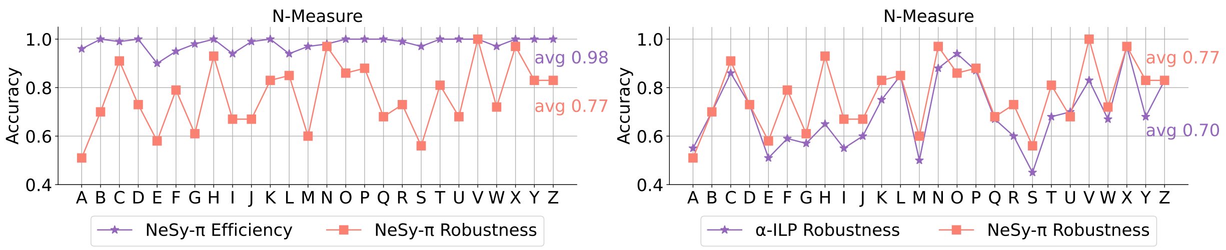


Fig. 11. **NeSy-π is robust to unseen falsifying examples.** The evaluation of Alphabet pattern's robustness is presented. **Left**: The NeSy-π performance in distinguishing negative examples is evaluated on two test sets. The purple line is based on a test set with dissimilar patterns as the negative examples, where different letters are used as negative examples. The orange line use a test set with similar patterns as negative examples, where one object is randomly removed from the same letters used as negative examples. **Right**: The performance of αILP and NeSy-π on the test set, where negative examples using similar patterns as positive patterns. The N-Measure of NeSy-π achieves a higher average score compared to αILP.

positive and negative examples using background knowledge and language biases. Many symbolic ILP frameworks have been developed, e.g., FOIL [14], Progol [13], ILASP [15], Metagol [16, 17], and Popper [18], showing their advantages of learning explanatory programs from small data. To handle uncertainties in ILP, statistical ILP approaches have been addressed, *e.g.* Markov Logic Networks [19] with their structure-learning algorithm [20]. Predicate invention (PI) has been a long-standing problem for ILP and many methods have been developed [17, 21–26], and extended to the statistical ILP systems [27].

Recently, differentiable ILP frameworks have been developed to integrate DNNs with logic reasoning and learning [5, 6, 28, 29]. ∂ILP [5] is the first end-to-end differentiable ILP solver, where they learn rules by gradient descent.

∂ILP can perform predication invention to compose solutions with less inductive biases. ∂ILP-ST [28] extends it by incorporating efficient search techniques developed for symbolic ILP systems, enabling the system to handle more complex rules using function symbols. To this end, Neuro-Symbolic ILP systems that can handle complex visual scenes as inputs have been developed. αILP [6] performs differentiable structure learning on visual scenes, however, these extended differentiable ILP systems do not support predicate invention. NeSy-π extends this approach by providing the predicate invention algorithm, leading them to learn from less supervision and background knowledge. Many other neuro-symblic systems have been established, *e.g.* DeepProbLog [30], NeurASP [31], SLASH [32] for parameter learning and FFNSL [33] for structure learning. NeSy-π could be integrated to these systems to allow predicate invention from complex visual scenes.

Meta rules, which define a template for rules to be generated, have been used for dealing with new predicates in (NeSy) ILP systems [5, 34]. NeSy-π achieves memory-efficient predicate invention system by performing scoring and pruning of candidates from given data, and this is crucial to handle complex visual scenes in NeSy-ILP systems since they are are memory-intensive [5].

Visual Reasoning. The machine-learning community has developed many visual datasets with reasoning requirements. Visual Question Answering (VQA) [35–37] is a well-established scheme to learn to answer questions given as natural language sentences together with input images. VQA assumes the programs to compute answers are given as a question, however, NeSy-π learns the programs from scratch using positive and negative examples. Kandinsky Patterns [9] are proposed as an environment to generate images that contain abstract objects (*e.g.* red cube) to evaluate neural and neural-symbolic systems. In a similar vein, CLEVR-Hans [11] has been developed, where the task is to classify 3D CLEVR scenes [7] based on classification rules that are defined on high-level concepts of objects' attributes and their relations. These datasets can be used to evaluate various models on visual reasoning, however, they do not involve with predicate invention, thus it is not trivial to evaluate predicate invention systems using them. The proposed *3D Kandinsky Patterns* extends these studies by having predicate invention tasks at its core, and it is the first to evaluate neuro-symbolic systems in terms of the ability of predicate invention.

6. Conclusion

We proposed **Ne**ural-**Sy**mbolic **P**redicate **I**nvention (NeSy-π), which discovers useful relational concepts from complex visual scenes. NeSy-π can reduce the amount of required background knowledge or supplemental pre-training of neuro-symbolic ILP frameworks by providing invented predicates to them. Thus NeSy-π extends the applicability of neuro-symbolic ILP systems to wide range of tasks. Moreover, we developed *3D Kandinsky Patterns datasets*, which is the first environment to evaluate neuro-symbolic predicate invention systems on 3D visual scenes. In our experiments, we have shown that (i) NeSy-π can invent useful predicates from visual scenes improving the performance of NeSy-ILP systems, (ii) NeSy-π can solve complex patterns that cannot be solved by the state-of-the-art NeSy-ILP baseline, (iii) NeSy-π is computationally efficient, and (iv) NeSy-π is robust to unseen falsifying examples. Developing a differentiable approach to fine-tune the given parameters is a promising avenue for future work. Also taking an unsupervised approach for predicate invention is another interesting future direction.

References

[1] S.H. Muggleton, Inductive Logic Programming, *New Gener. Comput.* **8**(4) (1991), 295–318.

[2] S.-H. Nienhuys-Cheng, R.d. Wolf, J. Siekmann and J.G. Carbonell, *Foundations of Inductive Logic Programming*, 1997.

[3] A. Cropper, S. Dumancic, R. Evans and S.H. Muggleton, Inductive logic programming at 30, *Mach. Learn.* **111**(1) (2022), 147–172.

[4] S. Muggleton and W.L. Buntine, Machine Invention of First Order Predicates by Inverting Resolution, in: *Proceedings of the Fifth International Conference on Machine Learning*, ML'88, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988, pp. 339–352–. ISBN 0934613648.

[5] R. Evans and E. Grefenstette, Learning Explanatory Rules from Noisy Data, *J. Artif. Intell. Res.* **61** (2018), 1–64.

[6] H. Shindo, V. Pfanschilling, D.S. Dhami and K. Kersting, αILP: thinking visual scenes as differentiable logic programs, *Mach. Learn.* (2023).

[7] J. Johnson, B. Hariharan, L. Van Der Maaten, L. Fei-Fei, C. Lawrence Zitnick and R. Girshick, Clevr: A diagnostic dataset for compositional language and elementary visual reasoning, in: *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2017, pp. 2901–2910.

[8] 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.

[9] H. Müller and A. Holzinger, Kandinsky Patterns, *Artificial Intelligence* **300** (2021), 103546.

[10] K. He, G. Gkioxari, P. Dollár and R. Girshick, Mask R-CNN, in: *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2980–2988. doi:10.1109/ICCV.2017.322.

[11] 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)*, 2021, pp. 3619–3629.

[12] Unity Technologies, Unity. https://unity.com/.

[13] S. Muggleton, Inverse Entailment and Progol, *New Generation Computing, Special issue on Inductive Logic Programming* **13**(3–4) (1995), 245–286.

[14] J.R. Quinlan, Learning Logical Definitions from Relations, *Mach. Learn.* **5** (1990), 239–266.

[15] M. Law, A. Russo and K. Broda, Inductive Learning of Answer Set Programs, in: *Logics in Artificial Intelligence - 14th European Conference (JELIA)*, E. Fermé and J. Leite, eds, Lecture Notes in Computer Science, Vol. 8761, 2014, pp. 311–325.

[16] A. Cropper and S.H. Muggleton, Metagol System, 2016. https://github.com/metagol/metagol.

[17] A. Cropper, R. Morel and S. Muggleton, Learning higher-order logic programs, *Mach. Learn.* **109** (2019), 1289–1322.

[18] A. Cropper and R. Morel, Learning programs by learning from failures, *Mach. Learn.* **110**(4) (2021), 801–856.

[19] M. Richardson and P.M. Domingos, Markov logic networks, *Mach. Learn.* **62**(1–2) (2006), 107–136.

[20] S. Kok and P. Domingos, Learning the Structure of Markov Logic Networks, in: *International Conference on Machine Learning*, 2005.

[21] I. Stahl, Predicate invention in ILP — an overview, in: *Machine Learning: ECML-93*, P.B. Brazdil, ed., Springer Berlin Heidelberg, Berlin, Heidelberg, 1993, pp. 311–322.

[22] D. Athakravi, K. Broda and A. Russo, Predicate Invention in Inductive Logic Programming, in: *2012 Imperial College Computing Student Workshop*, OpenAccess Series in Informatics (OASIcs), Vol. 28, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2012, pp. 15–21.

[23] C. Hocquette and S.H. Muggleton, Complete Bottom-Up Predicate Invention in Meta-Interpretive Learning, in: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, (IJCAI)*, International Joint Conferences on Artificial Intelligence Organization, 2020, pp. 2312–2318.

[24] S. Kramer, Predicate Invention : A Comprehensive View 1, 2007.

[25] A. Cropper, R. Morel and S.H. Muggleton, Learning Higher-Order Programs through Predicate Invention, *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)* (2020), 13655–13658.

[26] A. Cropper and R. Morel, Predicate Invention by Learning From Failures, *arXiv Preprint:2104.14426* (2021).

[27] S. Kok and P.M. Domingos, Statistical predicate invention, in: *International Conference on Machine Learning (ICML)*, 2007.

[28] H. Shindo, M. Nishino and A. Yamamoto, Differentiable Inductive Logic Programming for Structured Examples, in: *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, 2021, pp. 5034–5041.

[29] H. Shindo, V. Pfanschilling, D.S. Dhami and K. Kersting, Learning Differentiable Logic Programs for Abstract Visual Reasoning, *arXiv Preprint: 2307.00928* (2023).

[30] R. Manhaeve, S. Dumančić, A. Kimmig, T. Demeester and L. De Raedt, Neural probabilistic logic programming in DeepProbLog, *Artif. Intell.* **298** (2021), 103504.

[31] Z. Yang, A. Ishay and J. Lee, NeurASP: Embracing Neural Networks into Answer Set Programming, in: *International Joint Conference on Artificial Intelligence (IJCAI)*, C. Bessiere, ed., 2020, pp. 1755–1762.

[32] 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.

[33] D. Cunnington, M. Law, J. Lobo and A. Russo, FFNSL: Feed-Forward Neural-Symbolic Learner, *Mach. Learn.* **112**(2) (2023), 515–569.

[34] T. Kaminski, T. Eiter and K. Inoue, Exploiting Answer Set Programming with External Sources for Meta-Interpretive Learning, *Theory and Practice of Logic Programming* **18**(3–4) (2018), 571–588–. doi:10.1017/S1471068418000261.

[35] 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.

[36] 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** (2017), 21–40.

[37] 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) (2017), 32–73.

[38] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser and I. Polosukhin, Attention is all you need, *Advances in neural information processing systems (NeurIPS)* **30** (2017).

Appendix A. Experiment Setting.

A.1. Spatial Neural Predicates

To represent the spatial relationships between two objects, we have developed two types of spatial neural predicates, i.e., $\texttt{rho}(\texttt{O1},\texttt{O2},\rho)$ and $\texttt{phi}(\texttt{O1},\texttt{O2},\phi)$, which are given as basic knowledge. They are used for new predicates invention in some patterns associated with spatial concepts.

$rho(O1, O2, \rho)$ The neural predicate $\texttt{rho}(\texttt{O1},\texttt{O2},\rho)$ describes the distance between object $O1$ and $O2$, where ρ is the measure. Let W denotes the maximum possible distance between $O1$ and $O2$. To symbolize the distance between $O1$ and $O2$, we divide W into N_ρ parts and denote each part by $\rho_0, ..., \rho_{N_\rho - 1}$ (as shown in Figure 12 right). Each measure $\rho_i, 0 \leqslant i \leqslant N_\rho - 1$ actually covers a range of distance. For example, if $N_\rho = 4$, then ρ_0 denotes $[0.00W, 0.25W]$, ρ_1 denotes $[0.25W, 0.50W]$, ρ_2 denotes $[0.50W, 0.75W]$, ρ_3 denotes $[0.75W, 1.00W]$. In the experiment, we assign each ρ_i as the middle value of its covered range to ensure it is a constant, *e.g.* $\rho_0 = 0.125W, \rho_1 = 0.375W, \rho_1 = 0.625W, \rho_1 = 0.875W$ for $N_\rho = 4$. The positions of the objects are rounded to the value of closest ρ_i.

$phi(O1, O2, \phi)$ The predicate $phi(O1, O2, \phi)$ describes the direction of $O2$ with respect to $O1$. Based on the polar coordinate system, we consider the first argument $O1$ as the original point, then the direction of $O2$ is in the range $[0, 360)$. We divide the range into N sections, each section is denoted by $\phi_1, ..., \phi_N$ (as shown in Fig. 12). In the experiment, we assign each $\phi_i, 0 \leqslant i \leqslant N - 1$ as the middle value of its covered range to ensure it is a constant. *e.g.* $\phi_0 = 45°, \phi_1 = 135°, \phi_2 = 225°$ and $\phi_3 = 315°$ for $N = 4$.

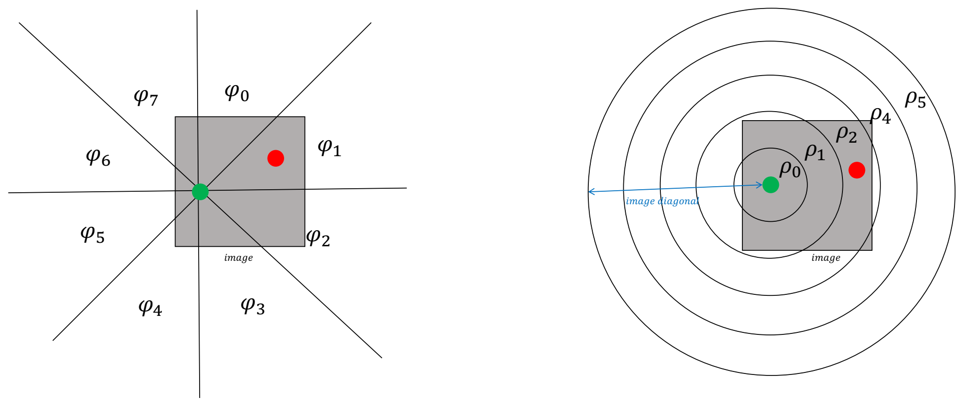


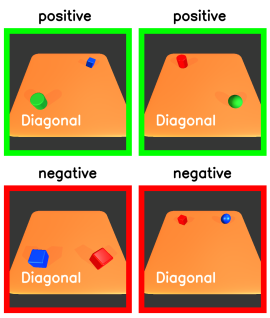
Fig. 12. Spatial neural predicates explanation by examples. **Left**: Directions of red circle with respect to green circle denoted by symbols $\phi_1, ..., \phi_7$. The red circle is in the direction of ϕ_1. **Right**: Distance of red circle to the green circle denoted by the symbols ρ_1,...,ρ_5. The red circle is located at the distance of ρ_2. For 3D KP, we assume all the object placed on a flat surface with same height. Thus their positions can be mapped to 2D space with ignoring the axis denoting the height. The positions of each object can be evaluated from depth map of 3D scenes.

.

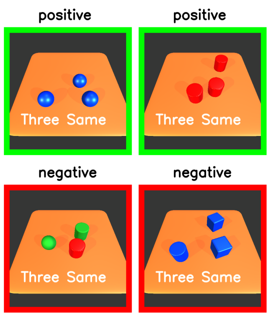
Appendix B. More Experiment Result

Figure 13 shows the learning result on Simple Kandinsky Patterns (Diagonal, Three Same, Two Pairs). Two positive and two negative images are given as examples in the left side. The target clauses searched by αILP and the predicated invented by NeSy-π are listed on the right side.

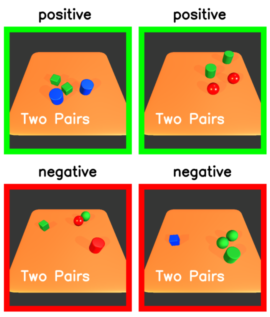
Figure. 15 shows the examples of each letter patterns in Alphabet Pattern Dataset. Each letter includes its upper and lower case examples. Figure. 17 shows the learning result on some of the Alphabet Patterns.



```
(Diagonal)
target(X):-in(O1,X),in(O2,X),inv_pred6(O1,O2).
inv_pred6(O1,O2):-in(O1,X),in(O2,X),phi(O1,O2,phi18).
inv_pred6(O1,O2):-in(O1,X),in(O2,X),phi(O1,O2,phi12).
inv_pred6(O1,O2):-in(O1,X),in(O2,X),phi(O1,O2,phi13).
(N_phi = 20)
```

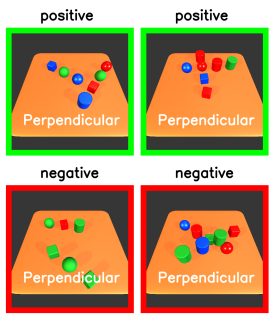


```
(Three Same)
target(X):-in(O1,X),in(O2,X),in(O3,X),inv_pred4(O1,O2,O3),
inv_pred61(O1,O2,O3).
inv_pred4(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),shape(O1,cube),
shape(O2,cube),shape(O3,cube).
inv_pred4(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),
shape(O1,cylinder),shape(O2,cylinder),shape(O3,cylinder).
inv_pred4(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),
shape(O1,sphere),shape(O2,sphere),shape(O3,sphere).
inv_pred61(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),
color(O1,green),color(O2,green),color(O3,green).
inv_pred61(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),
color(O1,blue),color(O2,blue),color(O3,blue).
inv_pred61(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),
color(O1,pink),color(O2,pink),color(O3,pink).
(Shapes: cube, sphere, cylinder; Colors: red, green, blue)
```

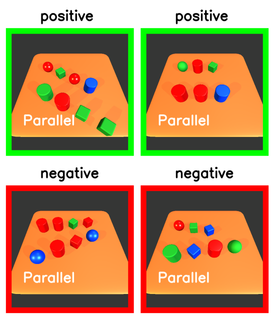


```
(Two Pairs)
target(X):-in(O1,X),in(O2,X),in(O3,X),in(O4,X),
inv_pred15(O1,O2,O3,O4),inv_pred158(O1,O2,O3,O4).
inv_pred15(O1,O2,O3,O4):-in(O1,X),in(O2,X),in(O3,X),in(O4,X),
color(O1,green),color(O2,green),color(O3,pink),color(O4,pink).
inv_pred15(O1,O2,O3,O4):-in(O1,X),in(O2,X),in(O3,X),in(O4,X),
color(O1,green),color(O2,green),color(O3,blue),color(O4,blue).
inv_pred15(O1,O2,O3,O4):-in(O1,X),in(O2,X),in(O3,X),in(O4,X),
color(O1,blue),color(O2,blue),color(O3,pink),color(O4,pink).
inv_pred158(O1,O2,O3,O4):-in(O1,X),in(O2,X),in(O3,X),in(O4,X),
shape(O1,cylinder),shape(O2,cylinder),shape(O3,cube),
shape(O4,cube).
inv_pred158(O1,O2,O3,O4):-in(O1,X),in(O2,X),in(O3,X),in(O4,X),
shape(O1,cube),shape(O2,cube),shape(O3,sphere),shape(O4,sphere).
inv_pred158(O1,O2,O3,O4):-in(O1,X),in(O2,X),in(O3,X),in(O4,X),
shape(O1,cylinder),shape(O2,cylinder),shape(O3,sphere),
shape(O4,sphere).
(Shapes: cube, sphere, cylinder; Colors: red, green, blue)
```

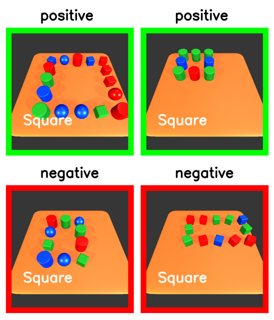
Fig. 13. **Simple Kandinsky Patterns** From top to bottom: Diagonal, Three Same, Two Pairs. In each patterns, two positive (green border) and two negative examples (red border) are shown in left side. The target clauses searched by NeSy-α and the predicates invented by NeSy-π are listed on the right side.



```
(Perpendicular)
target(X):-in(O1,X),in(O2,X),inv_pred102(O1),
inv_pred102(O2),inv_pred95(O1,O2),shape(O1,line).
inv_pred95(O1,O2):-in(O1,X),in(O2,X),rho(O1,O2,rho1).
inv_pred95(O1,O2):-in(O1,X),in(O2,X),rho(O1,O2,rho0).
inv_pred102(O1):-in(O1,X),slope(O1,slope0).
inv_pred102(O1):-in(O1,X),slope(O1,slope4).
inv_pred102(O1):-in(O1,X),slope(O1,slope1).
(N_slope = 5, N_rho = 5, N_phi = 5)
```



```
(Parallel)
target(X):-in(O1,X),in(O2,X),inv_pred0(O1,O2).
inv_pred0(O1,O2):-in(O1,X),in(O2,X),slope(O1,slope4),
slope(O2,slope4).
inv_pred0(O1,O2):-in(O1,X),in(O2,X),slope(O1,slope5),
slope(O2,slope5).
inv_pred0(O1,O2):-in(O1,X),in(O2,X),slope(O1,slope3),
slope(O2,slope3).
inv_pred0(O1,O2):-in(O1,X),in(O2,X),slope(O1,slope4),
slope(O2,slope5).
inv_pred0(O1,O2):-in(O1,X),in(O2,X),slope(O1,slope3),
slope(O2,slope2).
inv_pred0(O1,O2):-in(O1,X),in(O2,X),slope(O1,slope4),
slope(O2,slope3).
inv_pred0(O1,O2):-in(O1,X),in(O2,X),slope(O1,slope2),
slope(O2,slope2).
(N_slope = 8)
```



```
(Square)
target(X):-in(O1,X),in(O2,X),in(O3,X),inv_pred141(O1,O2,O3),
phi(O1,O2,phi3).
inv_pred141(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),
rho(O1,O2,rho6),rho(O1,O3,rho9),rho(O2,O3,rho6).
inv_pred141(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),
rho(O1,O2,rho7),rho(O1,O3,rho7),rho(O2,O3,rho10).
inv_pred141(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),
rho(O1,O2,rho5),rho(O1,O3,rho5),rho(O2,O3,rho7).
inv_pred141(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),
rho(O1,O2,rho6),rho(O1,O3,rho6),rho(O2,O3,rho8).
inv_pred141(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),
rho(O1,O2,rho5),rho(O1,O3,rho5),rho(O2,O3,rho8).
(N_rho = 15, N_phi = 8)
```

Fig. 14. **Complex Kandinsky Patterns** From top to bottom: Parallel, Perpendicular, Square. In each patterns, two positive (green border) and two negative examples (red border) are shown in left side. The target clauses searched by NeSy-α and the predicates invented by NeSy-π are listed on the right side.

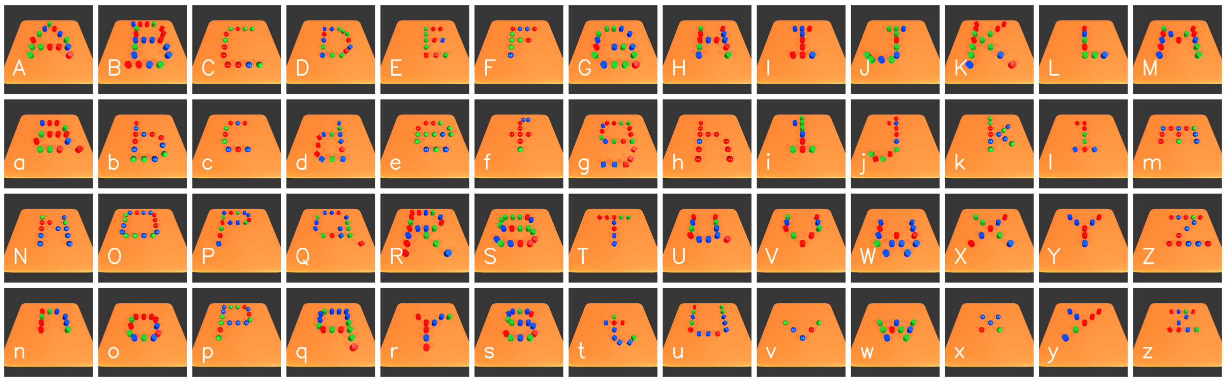
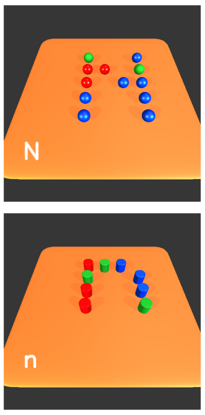

A B C D E F G H I J K L M
a b c d e f g h i j k l m
N O P Q R S T U V W X Y Z
n o p q r s t u v w x y z

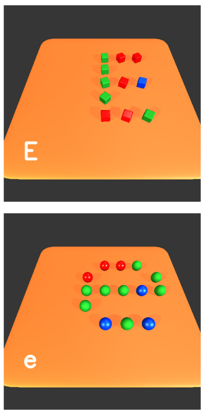
Fig. 15. **Latin Alphabet Patterns.** Latin letters (including both upper and lower cases) constructed by multiple objects. The shape of each letter is mainly composed of lines.



```
(Letter N)

target(X):-in(O1,X),in(O2,X),in(O3,X),inv_pred158(O1,O2,O3),
inv_pred649(O1,O2,O3),rho(O1,O2,rho9),shape(O1,line),
shape(O2,line),shape(O3,line),slope(O1,slope0),slope(O2,slope1).
inv_pred158(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),phi(O1,O2,phi5),
phi(O1,O3,phi4),phi(O2,O3,phi16).
inv_pred158(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),phi(O1,O2,phi18),
phi(O1,O3,phi16),phi(O2,O3,phi12).
inv_pred649(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),slope(O1,slope1),
slope(O2,slope0),slope(O3,slope3).
inv_pred649(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),slope(O1,slope1),
slope(O2,slope0),slope(O3,slope5).

N_slope = 10, N_rho = 20, N_phi = 20
```

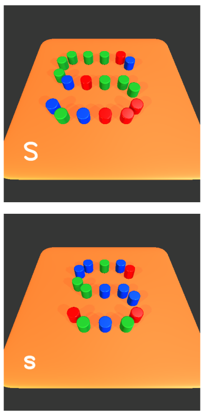


```
(Letter E)

target(X):-in(O1,X),in(O2,X),in(O3,X),inv_pred294(O1,O2,O3),
phi(O1,O3,phi5),shape(O1,line),shape(O2,line),shape(O3,line).
inv_pred294(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),rho(O2,O3,rho0).
inv_pred294(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),rho(O1,O3,rho3).
inv_pred294(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),rho(O1,O2,rho2).

N_slope = 10, N_rho = 20, N_phi = 20
```

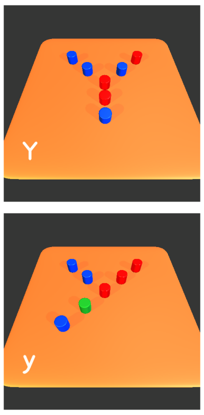
Fig. 16. **Alphabet Pattern N and E** From top to bottom: Letter N, E. In each patterns, upper and lower case examples are shown in left side. The target clauses searched by NeSy-α and the predicates invented by NeSy-π are listed on the right side.



```
(Letter S)

target(X):-in(O1,X),in(O2,X),inv_pred41(O1,O2),shape(O1,line),
shape(O2,line).
inv_pred41(O1,O2):-in(O1,X),in(O2,X),slope(O1,slope5),
slope(O2,slope5).
inv_pred41(O1,O2):-in(O1,X),in(O2,X),slope(O1,slope5),
slope(O2,slope4).
inv_pred41(O1,O2):-in(O1,X),in(O2,X),slope(O1,slope0),
slope(O2,slope0).
inv_pred41(O1,O2):-in(O1,X),in(O2,X),slope(O1,slope4),
slope(O2,slope2).

N_slope = 10, N_rho = 20, N_phi = 20
```



```
(Letter Y)

target(X):-in(O1,X),in(O2,X),in(O3,X),inv_pred284(O1,O2,O3),
inv_pred492(O2,O3).
inv_pred284(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),
rho(O1,O2,rho6),rho(O1,O3,rho7),rho(O2,O3,rho7).
inv_pred284(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),
rho(O1,O2,rho4),rho(O1,O3,rho0),rho(O2,O3,rho4).
inv_pred284(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),
rho(O1,O2,rho4),rho(O1,O3,rho6),rho(O2,O3,rho4).
inv_pred284(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),
rho(O1,O2,rho4),rho(O1,O3,rho9),rho(O2,O3,rho10).
inv_pred284(O1,O2,O3):-in(O1,X),in(O2,X),in(O3,X),
rho(O1,O2,rho4),rho(O1,O3,rho9),rho(O2,O3,rho4).
inv_pred492(O1,O2):-in(O1,X),in(O2,X),slope(O1,slope7),
slope(O2,slope3).
inv_pred492(O1,O2):-in(O1,X),in(O2,X),slope(O1,slope3),
slope(O1,slope7).

N_slope = 10, N_rho = 20, N_phi = 20
```

Fig. 17. **Alphabet Pattern S and Y** From top to bottom: Letter S, Y. In each patterns, upper and lower case examples are shown in left side. The target clauses searched by NeSy-α and the predicates invented by NeSy-π are listed on the right side.