

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

Jingyuan Sha^{1,*}, Hikaru Shindo¹, Kristian Kersting^{1,2,3} and Devendra Singh Dhami^{1,2}

¹Technische Universität Darmstadt

²Hessian Center for Artificial Intelligence (hessian.AI)

³German Research Centre for Artificial Intelligence (DFKI)

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 NeSy ILP systems that can handle visual input to learn logical rules using differentiable reasoning is relatively unaddressed. To this end, we propose a neural-symbolic approach, NeSy- π , to invent predicates from visual scenes for NeSy ILP systems based on clustering and extension of relational concepts. (π denotes the abbreviation 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 systems instead of hand-crafted background knowledge. Our experiments show that the PI model is capable of inventing high-level concepts and solving complex visual logic patterns more 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 based on given knowledge.

Keywords

Predicate Invention, Inductive Logic Programming, Neural Symbolic Artificial Intelligence

1. Introduction

Inductive Logic Programming (ILP) learns generalized logic programs from given data [1, 2, 3]. Unlike Deep Neural Networks (DNNs), ILP gains vital advantages, such as learning explanatory rules from small data. However, predicates for ILP systems are typically elusive and need to be hand-crafted, requiring much prior knowledge to compose solutions. Predicate invention (PI) systems invent new predicates that map new concepts from well designed primitive predicates given by experts, which extend the expression of the ILP language and consequently reduce the dependence on human experts [4]. A simple example

NeSy 2023, 17th International Workshop on Neural-Symbolic Learning and Reasoning, Certosa di Pontignano, Siena, Italy


*Corresponding author.

✉ jingyuan.sha@tu-darmstadt.de (J. Sha); hikaru.shindo@tu-darmstadt.de (H. Shindo);

kersting@cs.tu-darmstadt.de (K. Kersting); devendra.dhami@tu-darmstadt.de (D. S. Dhami)



© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

	NeSy-ILP	NeSy- π
Const.	Object:01/02, Shape:Sphere/Cube; Color:Blue/Pink/Green; Direction:Left/Right/Front/Behind	
Basic Pred.	color/2/obj,color;shape/2/obj,shape; dir/2/obj,obj	
B.K.	b_sp(01):-color(01,B),shape(01,Sp). g_sp(01):-color(01,G),shape(01,Sp). g_cu(01):-color(01,G),shape(01,Cu).	-
Pred.	left_side/2/obj,obj right_side/2/obj,obj	-

Figure 1: A logical pattern in 3D scenes can be learned by neural-symbolic ILP system. **Left:** Positive images in the first row, negative images in the second row. The pattern is: a blue sphere is either on the left side of a green sphere or on the right side of a green cube. **Right:** Comparison of language needs between NeSy-ILP and NeSy- π . Meaning of abbreviations in the table: Const-Constant. BK-Background Knowledge. Ne-Pred-Neural Predicate. B-Blue, G-Green, Sp-Sphere, Cu-Cube.

is the concept of the *blue sphere*. In classical NeSy-ILP systems, this can be explained by a clause $\text{blue_sphere}(X) : \text{-color}(X, \text{blue}), \text{shape}(X, \text{sphere})$ given as background knowledge. With PI systems, such concepts are learned from the basic predicates $\text{color}(X, \text{blue})$ and $\text{shape}(X, \text{sphere})$ by concatenation.

Recently, several neural-symbolic ILP frameworks have been proposed [5, 6] that incorporate DNNs for visual perception and learn explanation rules from raw inputs. NeSy-ILP systems outperform pure neural-based baselines on visual reasoning, where the answers are inferred by reasoning about objects’ attributes and their relations [7]. The major flaw of existing NeSy-ILP systems is that they require all predicates beforehand, *e.g.* pretrained neural predicates or explained by hand-crafted background knowledge. Moreover, the collection of background knowledge is very costly as it needs to be provided by human experts or requires pre-training of the neural modules with additional supervision. This severely limits the applicability of these NeSy-ILP systems to different domains. In contrast, DNNs require a minimal prior and achieve high performance by learning from data [8]. So the question arises: *How can we implement a NeSy-ILP system that can learn from less or no background knowledge?*

To this end, we propose Neural-Symbolic Predicate Invention (NeSy- π), a predicate invention pipeline that can invent relational concepts given visual scenes by using primitive predicates, reducing dependence on prior knowledge (see Fig. 1 for an example). NeSy- π discovers predicates describing attributes of objects and their spatial relations in visual scenes, which is equivalent to summarizing the required knowledge from scenes. The NeSy- π system (shown in Fig. 2) consists of two iterative steps: 1) evaluating clauses consisting of existing predicates and 2) inventing new predicates. To evaluate the existing predicates we propose two novel indicators of *necessity* and *sufficiency* which are computed by mapping the examples to a 2D Euclidean space. Only the predicates satisfying the above properties are considered in the invention of

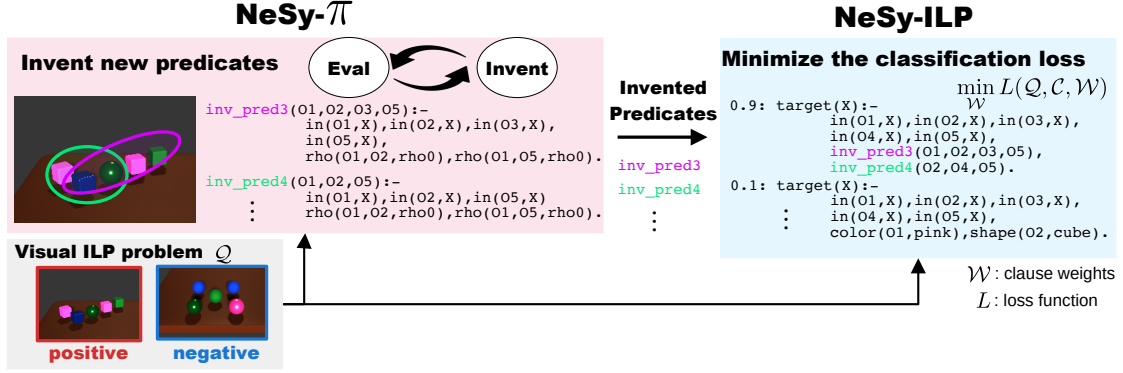


Figure 2: NeSy- π architecture. NeSy- π invents relational concepts for visual scenes by iterating *evaluation* and *invention* from primitive predicates and visual scenes (left). The invented predicates are fed into a NeSy-ILP solver to learn classification rules. The NeSy-ILP solver generates rule candidates using given predicates, performs differentiable reasoning using weighted clauses, and optimizes the clause weights by gradient descent. To this end, the classification rules can be efficiently assembled using the invented predicates (right). (Best viewed in color)

new predicates. NeSy- π can be integrated with existing NeSy-ILP systems providing them with a rich vocabulary to generate efficient solutions.

Overall, we make the following important contributions: **(1)** We propose NeSy- π , a NeSy predicate invention framework compatible with NeSy ILP systems. NeSy- π extends NeSy ILP systems by providing the capability to enrich their vocabularies by learning from data. **(2)** We propose two criteria for evaluating clauses, which can then be used for predicate invention tasks. **(3)** We develop 3D Kandinsky Patterns, which extends Kandinsky Patterns to the 3D world, and it achieves faster generation than other environments, *e.g.* CLEVR [9].

2. 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 p by p/n . An *atom* is a formula $p(t_1, \dots, t_n)$, where p is an n -ary predicate symbol and t_1, \dots, t_n are terms. A *ground atom* or simply a *fact* is an atom with no variables. A *literal* is an atom or its negation. A *positive literal* is simply an atom. A *negative literal* is the negation of an atom. A *clause* is a finite disjunction (\vee) of literals. A *ground clause* is a clause with no variables. A *definite clause* is a clause with exactly one positive literal. If A, B_1, \dots, B_n are atoms, then $A \vee \neg B_1 \vee \dots \vee \neg B_n$ is a definite clause. We write definite clauses in the form of $A :- B_1, \dots, B_n$. The atom A is called the *head*, and the set of negative atoms $\{B_1, \dots, B_n\}$ is called the *body*. For simplicity, we refer to the definite clauses as clauses in this paper. $\mathcal{I}_{\mathcal{F}}$ is the assignments of a function from \mathcal{D}^n to \mathcal{D} for each n -ary function

Inductive Logic Programming. ILP problem Q is tuple $(\mathcal{E}^+, \mathcal{E}^-, \mathcal{B}, \mathcal{L})$, where \mathcal{E}^+ is a set

of positive examples, \mathcal{E}^- 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) $\forall A \in \mathcal{E}^+, \mathcal{H} \cup \mathcal{B} \models A$ and (2) $\forall A \in \mathcal{E}^- \mathcal{H} \cup \mathcal{B} \not\models A$. Typically the search algorithm starts with general clauses. If the current clauses are too general (strong), i.e., they entail too many negative examples, then the solver specifies (weakens) them incrementally. This weakening operation is called a *refinement*, which is one of the essential tools for ILP.

Ne-Sy Inductive Logic Programming. We address the ILP problem in visual scenes, which is called *visual ILP problem*, where each example is given as an image with multiple objects [6]. The classification pattern is defined on high-level concepts such as attributes and relations of objects. To solve visual ILP problems, differentiable ILP frameworks have been proposed, e.g. ∂ ILP [5] and α ILP [6]. They use a differentiable implementation of *forward reasoning* in FOL and so that the classification rules can be learned by gradient descent, they solve $\min_{\mathcal{W}} \text{loss}(\mathcal{Q}, \mathcal{C}, \mathcal{W})$, where \mathcal{Q} is an ILP problem, \mathcal{C} is a set of clause candidates, \mathcal{W} is a set of clause weights, and *loss* is a loss function that returns a penalty when training constraints are violated. We note that we solve visual ILP problems where each positive and negative example is an image containing multiple objects.

3. Neuro-Symbolic Predicate Invention: NeSy- π

We propose NeSy- π , which invents new predicates for the NeSy-ILP solver from the dataset. Given initial language \mathcal{L} , initial clauses \mathcal{C}_0 , and dataset \mathcal{D} , NeSy- π aims to extend the language \mathcal{L} by inventing new predicates that make NeSy-ILP systems solve ILP problems efficiently. Algorithm 1 shows the PI algorithm of NeSy- π , and we describe each step in detail.

Clause Evaluation. We evaluate each clause to invent new predicates. **(Line 1–3)** A set of clauses \mathcal{C} is initialized, and extended by generating new clauses by adding a body atom to each clause in \mathcal{C} using the available predicates in \mathcal{L} . **(Line 5–7)** Promising predicates are those that can form a clause containing many positive examples but no negative examples. We develop *two* evaluation metrics for clauses in NeSy-ILP systems, which we call *necessity* and *sufficiency*. Intuitively, *necessary* clauses are those that are true in all positive examples, but they can also be true in negative examples. *Sufficient* clauses are those that are true in only some of positive examples, some

of positive examples maybe false, but they are always false in negative examples. To handle complex visual scenes, NeSy- π uses a differentiable reasoning function $r_{\mathcal{C}}$ for scenes from complex visual patterns in α ILP [6]. It computes logical entailment softly on visual scenes to

Algorithm 1 NeSy- π

Input: $\mathcal{L}, \mathcal{C}_0, \mathcal{D}$

```

1:  $\mathcal{C} \leftarrow \mathcal{C}_0$ 
2: while  $t < N$  do
3:    $\mathcal{C} \leftarrow \text{extend}(\mathcal{C}, \mathcal{L})$ 
4:   # Evaluate each set of clauses
5:   for  $\mathcal{C}_p \in \wp(\mathcal{C})$  do
6:      $s_{ness}^p \leftarrow f_{ness}(\mathcal{C}_p, \mathcal{D})$ 
7:      $s_{suff}^p \leftarrow f_{suff}(\mathcal{C}_p, \mathcal{D})$ 
8:   end for
9:   # Invent new predicates
10:   $\mathcal{C}_{ness}^* \leftarrow \text{top}_k(\wp(\mathcal{C}), \mathcal{S}_{ness})$ 
11:   $\mathcal{C}_{suff}^* \leftarrow \text{top}_k(\wp(\mathcal{C}), \mathcal{S}_{suff})$ 
12:   $\mathcal{L} \leftarrow \text{update}(\mathcal{L}, \mathcal{C}_{ness}^*, \mathcal{C}_{suff}^*)$ 
13: end while
14: return  $\mathcal{L}, \mathcal{C}$ 

```

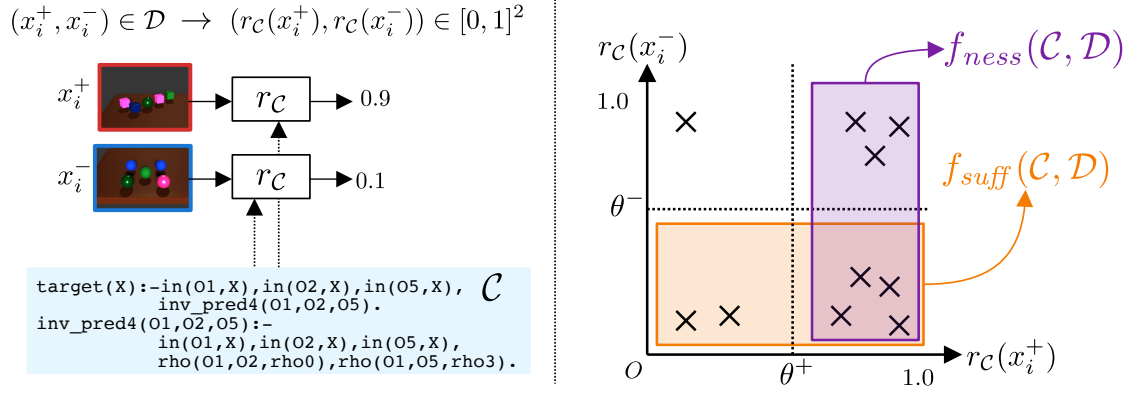


Figure 3: Scoring strategy in NeSy- π . NeSy- π takes a pair of positive and negative example (x_i^+, x_i^-) as input and applies the reasoning function $r_{\mathcal{C}}$ using clauses \mathcal{C} to compute a scalar value for each example. $r_{\mathcal{C}}$ computes the probability of classification label by differentiable forward reasoning using clauses \mathcal{C} in a visual scene (left). The computed pair of scalar values is transferred to a 2D Euclidean scoring space. f_{suff} quantifies the sufficiency of the clauses \mathcal{C} for the dataset \mathcal{D} , i.e. it counts the number of training pairs in \mathcal{D} whose negative example is not entailed by \mathcal{C} , and f_{ness} quantifies the necessity of the clauses \mathcal{C} for the dataset \mathcal{D} , i.e. it counts the number of training pairs in \mathcal{D} whose positive example is entailed by \mathcal{C} (right).

perform classification as follows: (i) conversion of a positive or negative visual scene x_i into a set of probabilistic atoms, (ii) differentiable forward reasoning using weighted clauses and probabilistic atoms, and (iii) the reasoning result is used to predict the label of the visual scene.

Given set of clauses \mathcal{C} , we assume that each subset of \mathcal{C} defines a new predicate, i.e. we consider $\mathcal{C}_p \in \wp(\mathcal{C})$ defining predicate p where $\wp(\mathcal{C})$ is a powerset of \mathcal{C} . To efficiently evaluate clauses, NeSy- π maps each training pair $(x_i^+, x_i^-) \in \mathcal{D}$, to a point in a 2D Euclidean scoring space $(r_{\mathcal{C}}(x_i^+), r_{\mathcal{C}}(x_i^-)) \in [0, 1]^2$, where $r_{\mathcal{C}}$ is a reasoning function for clauses \mathcal{C} , as shown in Fig. 3. $r_{\mathcal{C}}(x_i^+)$ represents the score of positive example x_i^+ , i.e., the probability that a positive example is classified as positive. $r_{\mathcal{C}}(x_i^-)$ represents the score of negative example x_i^- , i.e., the probability that a negative example is classified as positive. $|\mathcal{D}_+|$ and $|\mathcal{D}_-|$ represent the number of positive and negative images in the dataset, respectively. On the scoring space, the *necessity* of clauses \mathcal{C}_p is measured as follows:

$$s_{ness}^p = f_{ness}(\mathcal{C}_p, \mathcal{D}_+) = \frac{1}{|\mathcal{D}_+|} \sum_{x_i^+ \in \mathcal{D}_+} r_{\mathcal{C}_p}(x_i^+), \quad (1)$$

The *sufficiency* of clauses \mathcal{C}_p is measured as:

$$s_{suff}^p = f_{suff}(\mathcal{C}_p, \mathcal{D}_-) = \frac{1}{|\mathcal{D}_-|} \sum_{x_i^- \in \mathcal{D}_-} (1 - r_{\mathcal{C}_p}(x_i^-)), \quad (2)$$

Predicate Invention. NeSy- π invents new predicates by composing clauses to define new predicates using high scoring clauses in the two metrics. Given $\wp(\mathcal{C})$ and their evaluation

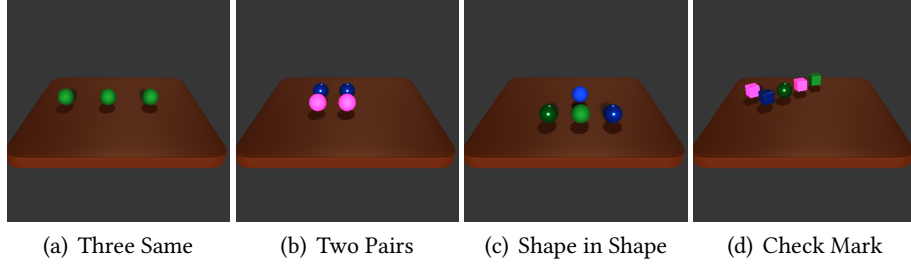


Figure 4: 3D Kandinsky Patterns. **Three Same:** This pattern always have 3 objects inside, three of them has same color and same shape. **Two Pairs:** This pattern always have 4 objects inside, two of them has same color and same shape. The other two also have same color and same shape. **Shape in Shape:** This pattern always have 4 objects with same shape inside, if they are cubes, their positions form a shape of square; if they are spheres, their positions form a shape of triangle. **Check Mark:** This pattern always have 5 objects, the position of 5 objects consists of an outline of a check mark. (For their varieties and false patterns please see Appendix A).

scores, \mathcal{S}_{ness} and \mathcal{S}_{suff} , top- k sets of clauses for each score are selected from $\wp(\mathcal{C})$, i.e. $\mathcal{C}_{ness}^* = top_k(\wp(\mathcal{C}), \mathcal{S}_{ness})$ and $\mathcal{C}_{suff}^* = top_k(\wp(\mathcal{C}), \mathcal{S}_{suff})$. Let $\mathcal{C}^* = \mathcal{C}_{ness}^* \cup \mathcal{C}_{suff}^*$, i.e. \mathcal{C}^* is a set of clause sets that have high scores in the metrics. New predicates are composed by taking *disjunction* of high-scored clauses to define them. Let $\{X_1, \dots, X_n\} \in \mathcal{C}^*$, we compose the following clauses to define a new predicate: $X_{new} \leftarrow body(X_1), \dots, X_{new} \leftarrow body(X_n)$, where X_{new} is an atom using a new predicate and $body(X)$ is body atoms of clause X .

For example, assume the following clauses defining `inv_pred1` and `inv_pred2` are selected:

```
inv_pred1(X):-in(01,X),in(02,X),color(01,blue),color(02,blue).
inv_pred2(X):-in(01,X),in(02,X),color(01,red),color(02,red).
```

then NeSy- π invents a new predicate by taking the disjunction of their bodies:

```
inv_pred3(X):-in(01,X),in(02,X),color(01,blue),color(02,blue).
inv_pred3(X):-in(01,X),in(02,X),color(01,red),color(02,red).
```

where `inv_pred3` is an invented 1-ary predicate that can be interpreted as: “*There is a pair of objects with the same color (blue or red) in the image.*”. By iterating the steps *evaluate* and *invent*, NeSy- π discovers new high-level relational concepts useful for classifying given visual scenes. The invented predicates are then fed into a NeSy ILP system to solve visual ILP problems.

4. Experimental Evaluation

We aim to answer the following questions: **Q1:** Does NeSy- π invent predicates from complex visual scenes for NeSy ILP systems, reducing required background knowledge? **Q2:** Is NeSy- π computationally efficient?

(A1. Predicate Invention) : We designed four visual scenes with different patterns for 2D and 3D Kandinsky patterns, respectively. (see Fig. 4). Each pattern has at least two varieties. For example, the pattern *three same* can be three same blue sphere, or three same pink cube, or something similar. All the patterns that we designed cannot be simply described by one or few given basic predicate, but need several long clauses. The results are compared with no predicate invention equipped model α ILP [6].

To solve such patterns, classical NeSy-ILP systems requires background knowledge and predicates to help the system reason about the final rules. For example, to solve pattern *three same*, NeSy-ILP can give the concept `two_same(A,B)` as background knowledge, then chains `two_same(A,B)` twice, i.e. `three_same(A,B,C) : -two_same(A,B), two_same(B,C).`, and `two_same(A,B)` is given as background knowledge described by other five BK predicates and five BK clauses as follows:

```
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,sq), shape(B,sq), in(A,X), in(B,X).
same_shape_pair(A,B) :- shape(A,cir), shape(B,cir), 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. The concepts are invented as new predicates. We only give six kinds of basic predicates as background knowledge. They are: `in/2/0,X`: evaluate if the object 0 exists in the image X. `shape/2/0,S`: evaluate if the object 0 has shape S. `color/2/0,C`: evaluate if the object 0 has color C. `phi/2/01,02, ϕ` : evaluate if the object 0₂ is located in the direction ϕ of 0₁ as calculated in polar coordinate system. `rho/2/01,02, ρ` : evaluate if the distance between object 0₂ and 0₁ is around ρ (For more details, please check Appendix A). A possible solution of pattern *three same* from NeSy- π can be learned as follows

```
target(A,B) :- in(01,X), in(02,X), inv_pred2(01,02).
inv_pred1(01,02) :- color(01,blue), color(02,blue), in(01,X), in(02,X).
inv_pred1(01,02) :- color(01,red), color(02,red), in(01,X), in(02,X).
inv_pred2(01,02) :- in(01,X), in(02,X), inv_pred1(01,02), shape(01,cir), shape(02,cir).
inv_pred2(01,02) :- in(01,X), in(02,X), inv_pred1(01,02), shape(01,sq), shape(02,sq).
```

In this case, the predicate `target(A,B)` corresponds the concept `two_same(A,B)` and it takes five clauses to explain it, which are learned instead of given. The predicates `inv_pred1(A,B)` and `inv_pred2(A,B)` are invented predicates. The other three predicates (`shape(A,Y)`, `color(A,Z)`, `in(A,X)`) are given beforehand. For more examples, please check Appendix. A.

The result of the evaluation of each pattern is shown in Tab. 1. For each pattern, we give the system only basic predicates and see that our system can successfully invent new predicates for complex visual patterns, resulting in improved accuracy. For most of the patterns, NeSy- π successfully finds the target rules by inventing new predicates and using them to describe the target patterns, while α ILP can only use existing predicates to describe them and fails at finding the correct target clauses. However, since the variety of each pattern is controlled in a small scale (limited choice of colors, shapes and positions), it is possible to find the target clauses that describe the pattern comprehensively, which also explains the reason for the accuracy of 1.0. The pattern *full house* has many more varieties (3024 varieties for positive patterns). Therefore

Dataset	Patterns	# obj	Iter.		Time (minutes)		Accuracy		# Preds	
			α	π	α	π	α	π	α	π
2D KP	Red Triangle	2	5	3	9.18	5.58	0.83	1.00	16	16 (+2)
	Two Pairs	4	8	4	11.27	8.91	0.64	1.00	4	4 (+5)
	Full House	4	8	8	14.79	4.12	0.83	0.90	4	4(+8)
	Check Mark	5	5	3	22.35	10.32	0.73	1.00	22	22 (+4)
3D KP	Same	3	5	3	0.91	0.72	0.55	1.00	4	4 (+3)
	Two Pairs	4	5	3	4.55	1.53	0.64	1.00	4	4 (+5)
	Shape of Shape	4	4	2	14.69	3.17	0.79	1.00	11	11 (+2)
	Check Mark	5	4	2	19.36	10.64	0.87	1.00	12	12 (+1)

Table 1

Experiment result on 2D Kandinsky Patterns and 3D Kandinsky Patterns. The dataset of each experiment has 64 PN pairs. α and π denote α ILP system (without background knowledge) and NeSy- π respectively. The brackets in last column represent the number of invented predicates used in the target clauses.

it is impossible to consider every single case, which is very time consuming. Therefore, no predicate was generated to describe it comprehensively. Nevertheless, the PI module improves the accuracy up to 0.9, which proves that the invented predicates improve the accuracy in incomplete description.

(A2. Computational Efficiency) : The runtime of NeSy- π depends mainly on the number of iterations needed to find the target clause. In each iteration, a predicate is added to each clause to obtain more complex rules. The higher the number of iterations, the more complex the rule becomes. Tab. 1 shows the time comparison of NeSy- π with α ILP (without background knowledge). Since the invented predicates entail the previously learned clauses, they are far more informative than the clauses learned by classical NeSy systems like α ILP. Therefore, NeSy- π is able to converge to the target clause faster, making it more efficient.

5. Related Work

Inductive Logic Programming [1, 10, 2, 3] has emerged at the intersection of machine learning and logic programming. Many ILP frameworks have been developed, e.g., FOIL [11], Progol [10], ILASP [12], Metagol [13, 14], and Popper [15]. Recently, NeSy ILP (Differentiable ILP) frameworks have been developed [5, 16, 6] to deal with visual inputs in ILP. Predicate invention (PI) has been a long-standing problem for ILP and many methods have been developed [17, 18, 14, 19, 20, 21, 22, 23]. Although these methods can invent predicates for symbolic inputs, predicate invention for Ne-Sy ILP systems has not been addressed. NeSy- π invents predicates given visual scenes using DNNs as a perception model, and to this end NeSy-ILP systems achieve better performance with less background knowledge or pre-training of neural modules. Meta rules, which define a template for rules to be generated, have been used for dealing with new predicates in (NeSy) ILP systems [5, 24]. 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

memory-intensive [5].

6. Conclusion

We proposed an approach for **Neural-Symbolic Predicate Invention** (NeSy- π). NeSy- π is able to find the new knowledge and summarize it as new predicates, thus it requires less background knowledge for reasoning. We show that our NeSy- π model uses only basic neural predicates for finding target clauses, i.e., no further background knowledge is required. Compared to the NeSy-ILP system without PI module, our system provides significantly more accurate classifications. Extending NeSy- π to more complex problems, e.g., with more objects in the scene, is an interesting avenue for future work. Developing an efficient prune strategy for finding longer clauses is another interesting direction.

References

- [1] S. H. Muggleton, Inductive logic programming, *New Gener. Comput.* 8 (1991) 295–318.
- [2] S.-H. Nienhuys-Cheng, R. d. Wolf, J. Siekmann, J. G. Carbonell, *Foundations of Inductive Logic Programming*, 1997.
- [3] A. Cropper, S. Dumancic, R. Evans, S. H. Muggleton, Inductive logic programming at 30, *Mach. Learn.* 111 (2022) 147–172.
- [4] S. Muggleton, 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, p. 339–352.
- [5] R. Evans, E. Grefenstette, Learning explanatory rules from noisy data, *J. Artif. Intell. Res.* 61 (2018) 1–64.
- [6] H. Shindo, V. Pfanschilling, D. S. Dhimi, K. Kersting, *oilp: thinking visual scenes as differentiable logic programs*, *Mach. Learn.* (2023).
- [7] J. Mao, C. Gan, P. Kohli, J. B. Tenenbaum, J. Wu, The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision, in: *International Conference on Learning Representations (ICLR)*, 2019.
- [8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, *Advances in neural information processing systems (NeurIPS)* 30 (2017).
- [9] J. Johnson, B. Hariharan, L. Van Der Maaten, L. Fei-Fei, C. Lawrence Zitnick, 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.
- [10] S. Muggleton, Inverse Entailment and Progol, *New Generation Computing, Special issue on Inductive Logic Programming* 13 (1995) 245–286.
- [11] J. R. Quinlan, Learning logical definitions from relations, *Mach. Learn.* 5 (1990) 239–266.
- [12] M. Law, A. Russo, K. Broda, Inductive learning of answer set programs, in: E. Fermé, J. Leite (Eds.), *Logics in Artificial Intelligence - 14th European Conference (JELIA)*, volume 8761 of *Lecture Notes in Computer Science*, 2014, pp. 311–325.

- [13] A. Cropper, S. H. Muggleton, Metagol system, <https://github.com/metagol/metagol>, 2016. URL: <https://github.com/metagol/metagol>.
- [14] A. Cropper, R. Morel, S. Muggleton, Learning higher-order logic programs, *Mach. Learn.* 109 (2019) 1289 – 1322.
- [15] A. Cropper, R. Morel, Learning programs by learning from failures, *Mach. Learn.* 110 (2021) 801–856.
- [16] H. Shindo, M. Nishino, A. Yamamoto, Differentiable inductive logic programming for structured examples, in: *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, 2021, pp. 5034–5041.
- [17] I. Stahl, Predicate invention in ilp – an overview, in: P. B. Brazdil (Ed.), *Machine Learning: ECML-93*, Springer Berlin Heidelberg, Berlin, Heidelberg, 1993, pp. 311–322.
- [18] D. Athakravi, K. Broda, A. Russo, Predicate Invention in Inductive Logic Programming, in: *2012 Imperial College Computing Student Workshop*, volume 28 of *OpenAccess Series in Informatics (OASISs)*, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2012, pp. 15–21.
- [19] C. Hocquette, 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.
- [20] S. Kok, P. M. Domingos, Statistical predicate invention, in: *International Conference on Machine Learning (ICML)*, 2007.
- [21] S. Kramer, Predicate invention : A comprehensive view 1, 2007.
- [22] A. Cropper, R. Morel, S. H. Muggleton, Learning higher-order programs through predicate invention, *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI) (2020)* 13655–13658.
- [23] A. Cropper, R. Morel, Predicate invention by learning from failures, *arXiv Preprint:2104.14426* (2021).
- [24] T. KAMINSKI, T. EITER, K. INOUE, Exploiting answer set programming with external sources for meta-interpretive learning, *Theory and Practice of Logic Programming* 18 (2018) 571–588. doi:10.1017/S1471068418000261.

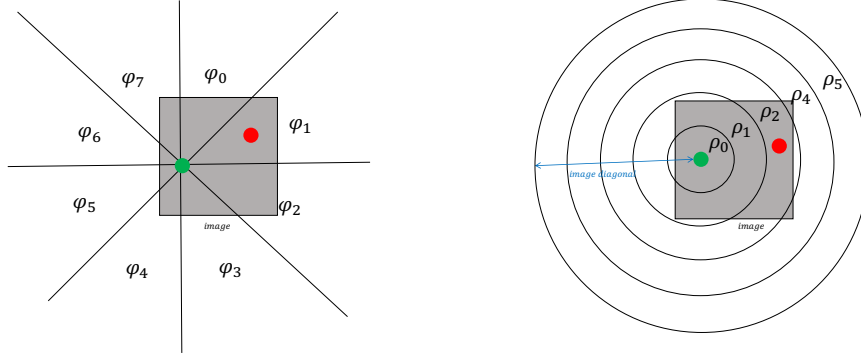


Figure 5: Spatial neural predicates explanation by examples. **Left:** Evaluation of the directions of red circle with respect to green circle using predicate ϕ_1, \dots, ϕ_7 . The red circle is in the direction of ϕ_1 . **Right:** Evaluation of distance of red circle to the green circle using the predicate ρ_1, \dots, 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 height axis. The positions of each object can be evaluated from depth map of 3D scenes.

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., $\text{rho}(O1, O2, \rho)$ and $\text{phi}(O1, O2, \phi)$, which are given as basic knowledge. They are used for new predicates invention in some patterns associated with spatial concepts.

$\text{rho}(O1, O2, \rho)$ The neural predicate $\text{rho}(O1, O2, \rho)$ describes the distance between object $O1$ and $O2$, where ρ is the measure. Let W denotes the length of image diagonal. Obviously, W is 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 $\rho_i, 1 \leq i \leq N_\rho$ (as shown in Fig. 5 right). For example, if $N_\rho = 4$, then $\rho_1 = 0.25W, \rho_2 = 0.5W, \rho_3 = 0.75W, \rho_4 = 1.0W$. The actual distance between two objects can be symbolized by ρ_1, ρ_2, ρ_3 or ρ_4 .

$\text{phi}(O1, O2, \phi)$ The predicate $\text{phi}(O1, O2, \phi)$ 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 symbolize all directions with ϕ_1, \dots, ϕ_N , where N is a hyper-parameter. The actual direction can be classified to one of these symbols (as shown in Fig. 5).

A.2. Visual Patterns

Fig. 6 illustrates the four training patterns in the 2D Kandinsky dataset. Fig. 7 illustrates the four training patterns in 3D Kandinsky dataset.

A.3. Invented Predicates

Fig. 8 shows the invented predicates and their associated clauses in the 3D Kandinsky pattern dataset.

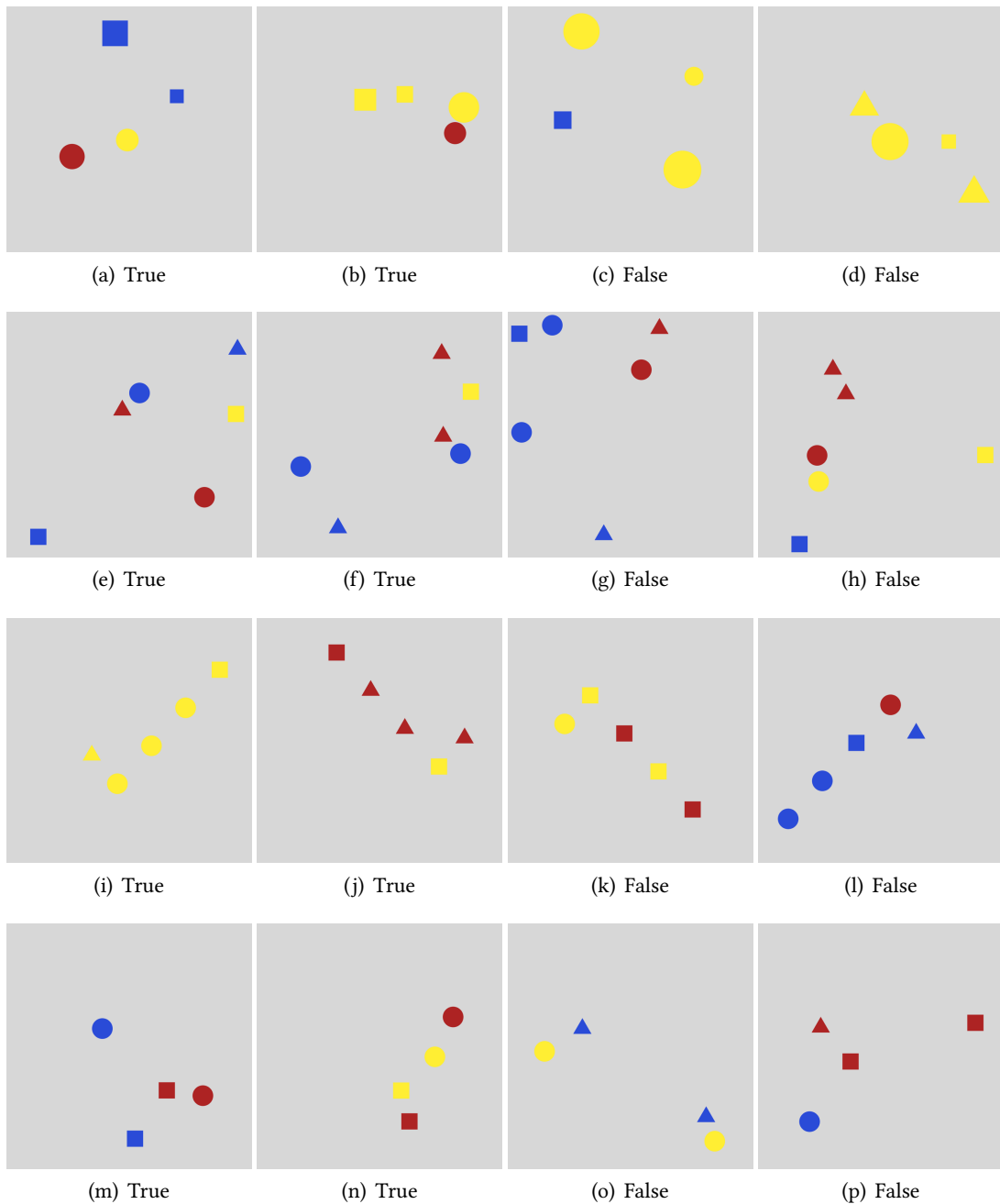


Figure 6: Kandinsky Patterns. **Two Pairs:** In this pattern there are always four objects, two of them have the same color and the same shape, the other two have the same shape and a different color. **Red Triangle:** This pattern always contains a red triangle and another object with different color and shape nearby, the remaining four objects are random. **Check Mark:** This pattern always consists of five objects whose positions form an outline of check mark. It includes 3 symmetrical patterns. We consider two of them with convex shapes as true and the other two with concave shape as false. **Full House:** In this pattern, there are always four different objects. i.e. none of two objects are exactly the same, they either have a different color or a different shape. In the false patterns, at least two objects are exactly the same, i.e., they have same color and the same shape. Actually, full house is the opposite case of one pair, the system failed on this pattern (with 0.90 accuracy). The reason is that there are too many different variants in the true patterns and the probability of each variant is low, so they are difficult to capture. To invent a predicate that includes all patterns, we need to select more clauses in each iteration and use them to invent a new predicate. But considering more clauses is very memory intensive. One solution may be to start with false patterns, which consists one pair, has less variety, and is therefore easy to describe. Then the truth pattern is the negation of the false pattern.

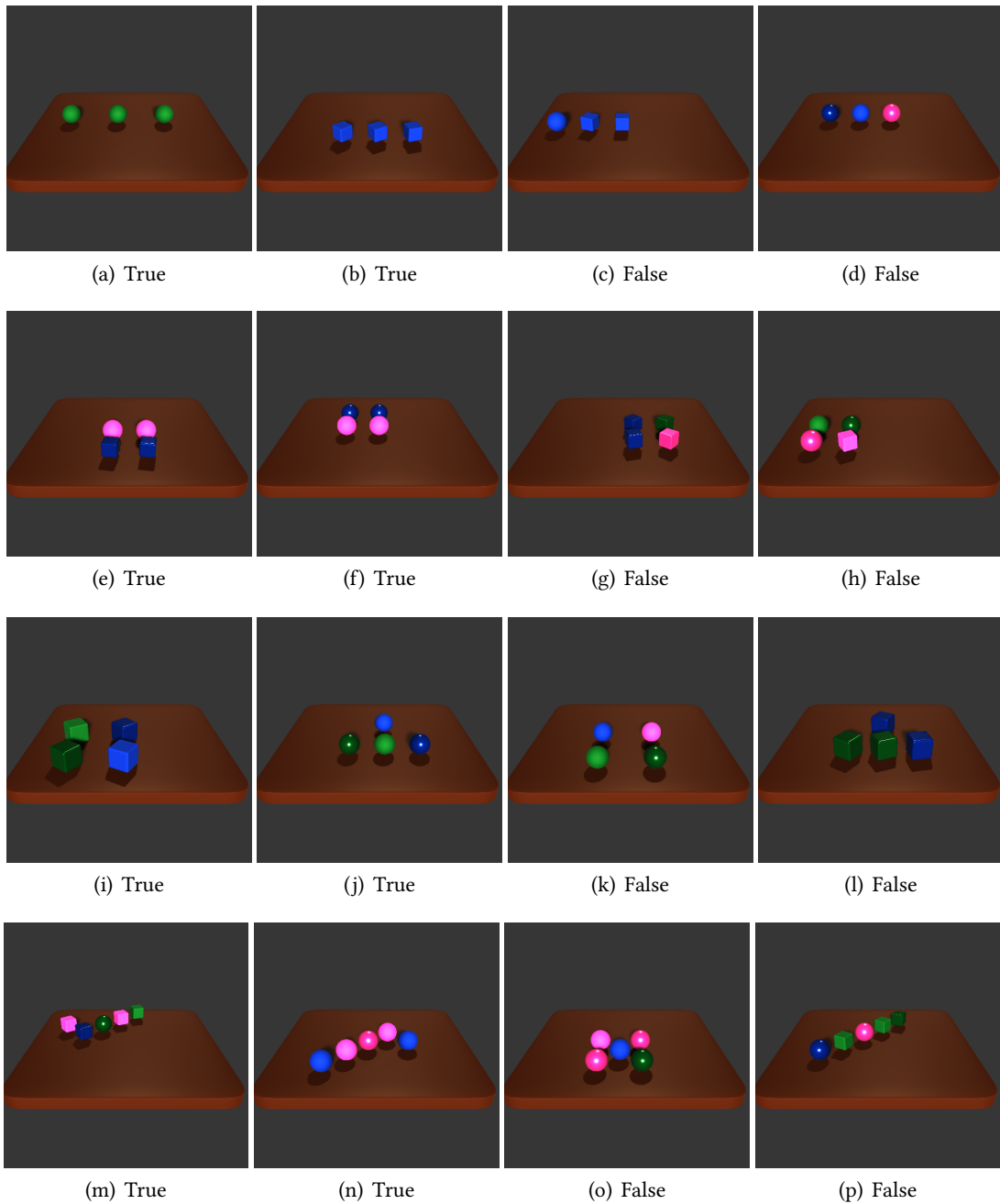


Figure 7: Training examples in 3D Kandinsky pattern dataset. From top to bottom, **Three Same:** Three objects with the same color and the same shape. **Two Pairs:** Four objects, two of which have the same color and shape, the other two also have the same color and shape. **Shape in Shape:** Four objects with the same shape. If all objects are cubes, their positions form a square shape. If all objects are spheres, their positions form a triangle shape. **Check Mark:** Five objects form a check mark shape, the mirrored or symmetrical shapes are also considered as positive.


```

(three same)
kp(X):-in(01,X),in(02,X),in(03,X),inv_pred100(01,02,03),inv_pred451(01,02,03).
inv_pred451(01,02,03):-in(01,X),in(02,X),in(03,X),inv_pred2(01,02),
inv_pred2(01,03),inv_pred2(02,03).
inv_pred100(01,02,03):-in(01,X),in(02,X),in(03,X),shape(01,cube),
shape(02,cube),shape(03,cube).
inv_pred100(01,02,03):-in(01,X),in(02,X),in(03,X),shape(01,sphere),
shape(02,sphere),shape(03,sphere).
inv_pred2(01,02):-color(01,blue),color(02,blue),in(01,X),in(02,X).
inv_pred2(01,02):-color(01,green),color(02,green),in(01,X),in(02,X).
inv_pred2(01,02):-color(01,pink),color(02,pink),in(01,X),in(02,X).

(two pairs)
kp(X):-in(01,X),in(02,X),in(03,X),in(04,X),inv_pred38(01,02,03,04),
inv_pred9(01,02),inv_pred9(03,04).
inv_pred38(01,02,03,04):-in(01,X),in(02,X),in(03,X),in(04,X),inv_pred2(03,04),
inv_pred3(01,02),inv_pred4(03,04).
inv_pred9(01,02):-in(01,X),in(02,X),shape(01,cube),shape(02,cube).
inv_pred9(01,02):-in(01,X),in(02,X),shape(01,sphere),shape(02,sphere).
inv_pred2(01,02):-color(01,blue),in(01,X),in(02,X),shape(01,sphere).
inv_pred2(01,02):-color(01,green),color(02,green),in(01,X),in(02,X).
inv_pred2(01,02):-color(01,pink),color(02,pink),in(01,X),in(02,X).
inv_pred3(01,02):-color(01,blue),color(02,blue),in(01,X),in(02,X).
inv_pred3(01,02):-color(01,green),color(02,green),in(01,X),in(02,X).
inv_pred3(01,02):-color(01,pink),color(02,pink),in(01,X),in(02,X).
inv_pred4(01,02):-color(01,blue),color(02,blue),in(01,X),in(02,X).
inv_pred4(01,02):-color(01,blue),in(01,X),in(02,X),shape(01,cube).
inv_pred4(01,02):-color(01,green),color(02,green),in(01,X),in(02,X).
inv_pred4(01,02):-color(01,pink),color(02,pink),in(01,X),in(02,X).

(shape in shape)
kp(X):-in(01,X),in(02,X),in(03,X),in(04,X),inv_pred7(01,02,03,04),
phi(02,04,phi2),rho(01,04,rho1).
inv_pred7(01,02,03,04):-in(01,X),in(02,X),in(03,X),in(04,X),inv_pred1(01,02,03),
phi(01,04,phi3).
inv_pred7(01,02,03,04):-in(01,X),in(02,X),in(03,X),in(04,X),inv_pred1(01,02,03),
shape(01,sphere).
inv_pred1(01,02,03):-in(01,X),in(02,X),in(03,X),rho(02,03,rho0).
inv_pred1(01,02,03):-in(01,X),in(02,X),in(03,X),shape(01,cube).

(check mark)
kp(X):-in(01,X),in(02,X),in(03,X),in(04,X),in(05,X),inv_pred3(02,03,04,05),
rho(02,05,rho0),rho(03,04,rho1).
inv_pred3(01,02,03,05):-in(01,X),in(02,X),in(03,X),in(05,X),phi(01,02,phi2),
phi(02,05,phi1).
inv_pred3(01,02,03,05):-in(01,X),in(02,X),in(03,X),in(05,X),phi(02,05,phi1),
phi(03,05,phi4).

```

Figure 8: Learned rules by NeSy- π on 3D Kandinsky Pattern scenes.