# DIFFERENTIABLE META-LOGICAL PROGRAMMING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Deep learning uses an increasing amount of computation and data to solve very specific problems. By stark contrast, human minds solve a wide range of problems using a fixed amount of computation and limited experience. One ability that seems crucial to this kind of general intelligence is meta-reasoning, i.e., our ability to reason about reasoning. To make deep learning do more from less, we propose the differentiable logical meta interpreter (DLMI). The key idea is to realize a meta-interpreter using differentiable forward-chaining reasoning in first-order logic. This directly allows DLMI to reason and even learn about its own operations. This is different from performing object-level deep reasoning and learning, which refers in some way to entities external to the system. In contrast, DLMI is able to reflect or introspect, i.e., to shift from meta-reasoning to object-level reasoning and vice versa. Among many other experimental evaluations, we illustrate this behavior using the novel task of "repairing Kandinsky patterns," i.e., how to edit the objects in an image so that it agrees with a given logical concept.

## 1 INTRODUCTION

One of the ultimate goals of artificial intelligence is to build a fully autonomous or 'human-like' system, which is also termed as artificial general intelligence (AGI). The current successful deep-learning systems such as DALLE-2 (Ramesh et al., 2022) and Gato (Reed et al., 2022) have been touted to take the field closer to this dream. Albeit, the main drawback of such deep-learning system is that they use numerous computations in order to solve very specific tasks. For example, DALLE-2 can generate very high quality images but cannot play chess or Atari games. By stark contrast, human minds solve a wide range of problems using a fixed amount of computation and limited experience. For a system to be really considered a step towards AGI, it has to be self-reflective, i.e., it should have the ability to learn from itself as well as reason about its own capabilities. In other words, the study of meta-level architectures such as meta-learning (Schmidhuber, 1987) and meta-reasoning (Griffiths et al., 2019b) becomes progressively important.

Meta learning in machine learning aims to achieve learning-to-learn (Thrun & Pratt, 1998), i.e., improve the learning algorithm itself (Hospedales et al., 2022; Finn et al., 2017) where the main focus is at learning on the meta-level. Meta-reasoning refers to the process of "reason about reasoning" and is in the same realm as meta-learning. This means that a system is able to reason about its own capabilities. This is different from performing object-level reasoning, which refers to entities external to the system. A system capable of meta-reasoning may be able to reflect, or introspect, i.e. to shift from meta-reasoning to object level reasoning and vice versa.

Although deep learning methods have positively impacted the march towards building 'human-like' systems, it still suffers from several problems such as large data requirement, being black box and not able to perform reasoning. The last drawback is particularly frustrating because of their considerable performance in the perception of sensory inputs. Recently, a lot of work has focused on the challenging task of object-centric reasoning, (Kim et al., 2018; Stammer et al., 2021; Shindo et al., 2021a) where the models perform low-level visual perception and reasoning on high-level concepts. Accordingly, there has been a push to make these reasoning systems differentiable (Evans & Grefenstette, 2018; Shindo et al., 2021b) along with proposing new benchmarks such as CLEVR (Johnson et al., 2017) and Kandinsky patterns (Holzinger et al., 2019; Müller & Holzinger, 2021). These differentiable reasoning systems use an object-centric neural network as a perception module to extract required information and the logical reasoner is on top of the neural network which is able to reason the output of the neural network. Although this can solve the proposed benchmarks to some extent, the critical question remains unanswered: *Is the reasoner able to learn its own operations?*

For example, consider the Kandinsky pattern consisting of 4 objects in Fig. 1 (left), where the task is to arrange the objects in a diagonal as in Fig. 1 (right), taking into consideration relations among the objects and also produce the intermediate steps of the reasoning to trace. The pattern behind of this figure is: *"The Kandinsky Figure has two pairs of objects with the same shape. In one pair, the objects have the same colors, in the other pair different colors. Two pairs are always disjunct, i.e., they do not share objects."*. Due to the lack of reasoning ability in complex scenes with complicated relations, it is difficult for classical deep networks to complete the given task. Moreover, due to the lack of the ability of the meta-reasoning, the previous differentiable reasoners are limited to produce the results simply without any proof sketch. The function to produce the proof-trees in parallel in the reasoning system can be realized by having the meta-level programs, i.e., programs defining the way to perform reasoning building proof-trees recursively in the middle of reasoning.

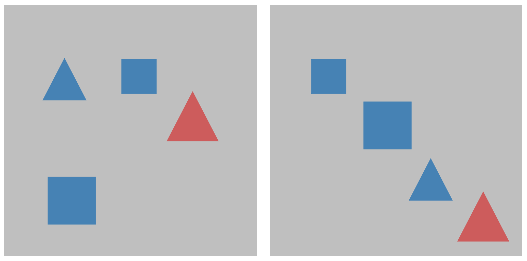


Figure 1: The left plot is a original Kandinsky pattern and the right plot shows one example of the novel task "repairing Kandinsky patterns".

This work takes a descriptive approach of the meta programming, i.e., aims to integrate the meta-level programming into the machine-learning paradigm. To this end, we propose a meta interpreter, *differentiable logical-meta interpreter (DLMI)*, that can reason or even learn about its own operations. The key idea is to realize the meta interpreter using differentiable forward-chaining reasoning in first-order logic. Among many other abilities, DLMI can answer the question: why has an algorithm made a particular decision? Moreover, we propose a novel differentiable planner, which we can robustly perform the novel task of "repairing Kandinsky patterns". The task is to edit the objects in an image so that it agrees with a given logical concept. Another advantage of our proposed logical-meta interpreter DLMI is that it can be modified to construct the proof tree, which is able to give proofs of the outputs, thus making it more trustworthy/explainable than existing differentiable reasoning techniques. Overall, we make the following important contributions:

1. We propose the first differentiable logical-meta interpreter (DLMI) that can reason about its own decisions.
2. We show that DLMI can be used to construct a proof tree for the underlying query, thus making it more explainable.
3. We realize a differentiable planner based on first-order logic via DLMI framework.
4. We propose a novel task of "repairing Kandinsky patterns" and show that DLMI is able to solve it efficiently.

To this end, we will start off by reviewing first-order logic and reasoning. Afterwards, we introduce differentiable logical meta programming and illustrate it before concluding.

## 2 First-Order Logic and Reasoning

In this section, we will briefly revisit first-order logic and reasoning.

**First-Order Logic.** A *term* is a constant, a variable, or a term which consists of a function symbol. We denote $n$-ary predicate $\mathtt{p}$ by $\mathtt{p}/(n, [\mathtt{dt_1}, \ldots, \mathtt{dt_n}])$, where $\mathtt{dt_i}$ is the datatype of $i$-th argument. An *atom* is a formula $\mathtt{p}(\mathtt{t_1}, \ldots, \mathtt{t_n})$, where $\mathtt{p}$ is an $n$-ary predicate symbol and $\mathtt{t_1}, \ldots, \mathtt{t_n}$ are terms. A *ground atom* or simply a *fact* is an atom with no variables. A *literal* is an atom or its negation. A *positive literal* is just an atom. A *negative literal* is the negation of an atom. A *clause* is a finite disjunction ($\lor$) 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 $A$ :- $B_1, \ldots, B_n$. Atom $A$ is called the *head*, and

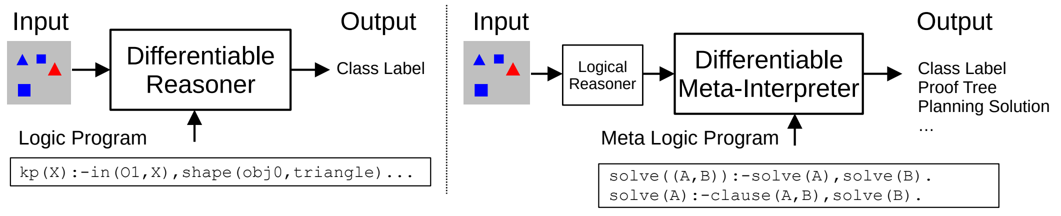


Figure 2: **DLMI vs existing differentiable reasoners**. In contrast to the conventional differentiable reasoner (left), DLMI takes meta-logic programs to reason, i.e., programs to interpret the programs. Besides providing the results of the original logical reasoner with equivalent accuracy, DLMI can realize many important functions in a differentiable way, e.g., holding the proof trees and solving planning problems (right).

a set of negative atoms $\{B_1, \ldots, B_n\}$ is called the *body*. We call definite clauses as clauses for the simplicity in this paper.

**Differentiable Forward-Chaining Reasoning.** The forward-chaining inference is a type of inference in first-order logic to compute logical entailment (Russell & Norvig, 2009). The differentiable forward-chaining inference (Evans & Grefenstette, 2018; Shindo et al., 2021b) computes the logical entailment in a differentiable manner. We briefly summarize the steps: **(Step 1)** A tensor that holds the relationships between clauses and facts is computed. **(Step 2)** Each clause is compiled into a differentiable function that performs forward reasoning using the tensor. **(Step 3)** A differentiable logic program is composed of the clause functions and their weights. $T$-time step inference is computed by amalgamating the inference results recursively.

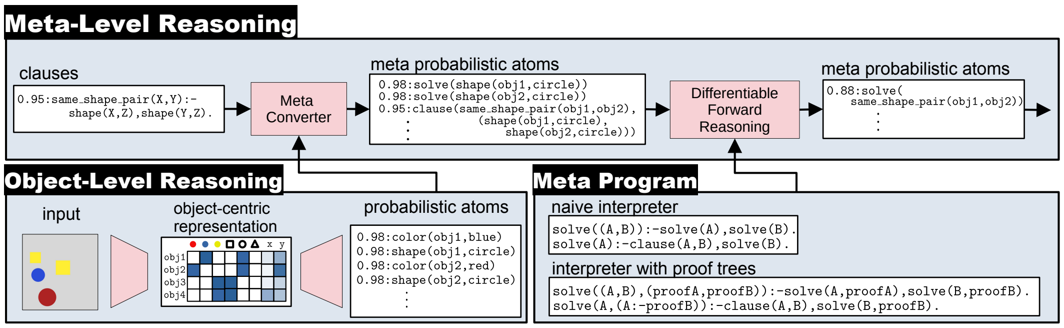
Differentiable forward-reasoning has also been applied to the visual domain. Neural Symbolic Forward Reasoner (NSFR) (Shindo et al., 2021a) is a neuro-symbolic framework, which combines the individual strengths of neural networks and logical reasoning and performs differentiable forward-chaining reasoning given visual inputs. The reasoning process is implemented using only simple tensor operations.

**Meta Reasoning and Learning.** Meta-reasoning is the study about the system which is able to reason about its operation, i.e., a system capable of meta-reasoning may be able to reflect, or introspect (Maes & Nardi, 1988), i.e., to shift from meta-reasoning to object-level reasoning and vice versa (Costantini, 2002; Griffiths et al., 2019a). Compared with imperative programming, it is relative easier to construct a meta interpreter using declarative programming. As an instance of declarative programming, logic programming paradigms, e.g., Prolog (Sterling & Shapiro, 1994) has a special ability of efficiently and compactly evaluating itself. First-order Logic (Lloyd, 1984) has been the major tool to realize the mata-reasoning systems (Hill & Gallagher, 1998; Pettorossi, 1992; Apt & Turini, 1995).

## 3 Differentiable Meta-Logical Programming

As an instance of meta programming, we begin this section by introducing the idea of meta interpreter. Then we describe in detail the structure of Differentiable Logical-Meta Interpreter (DLMI), which applies the idea of differentiable forward-chaining reasoning to realize the meta interpreter in a differentiable manner. The basic difference between object level reasoning and meta level reasoning can be seen in Fig. 2. Different from performing object-level deep reasoning and learning, meta level reasoning takes the logic algorithm as an input and reasons on the meta level, which grants the meta interpreter with initial benefits of realizing many important functions (such as proof tree) which are for other systems hard to realize.

Fig. 3 shows an overview of the DLMI structure. An object-level reasoning program serves as the input. Meta converter transforms the probabilistic atoms into meta probabilistic atoms, then the meta interpreter performs the differentiable forward reasoning using the meta rules. Before we introduce the individual components, we first define the meta predicates $solve$ and $clause$.



```
0.95:same_shape_pair(X,Y):-
        shape(X,Z),shape(Y,Z).
```

```
0.98:solve(shape(obj1,circle))
0.98:solve(shape(obj2,circle))
0.95:clause(same_shape_pair(obj1,obj2),
              (shape(obj1,circle),
                shape(obj2,circle)))
  :
```

```
0.88:solve(
    same_shape_pair(obj1,obj2))
    :
```

```
0.98:color(obj1,blue)
0.98:shape(obj1,circle)
0.98:color(obj2,red)
0.98:shape(obj2,circle)
      :
```

```
solve((A,B)):-solve(A),solve(B).
solve(A):-clause(A,B),solve(B).
```

```
solve((A,B),(proofA,proofB)):-solve(A,proofA),solve(B,proofB).
solve(A,(A:-proofB)):-clause(A,B),solve(B,proofB).
```

Figure 3: **Overview of the DLMI framework using differentiable forward chaining and implementing memory using functors**. DLMI takes a logic program as input and the outputs depend on the function choices of DLMI such as planning steps, proof tree etc. We provide two meta rules using functors in the meta program component. One is the rule for "naive interpreter" and the other is for "interpreter with proof tree". To switch from naive interpreter to interpreter with proof tree, we just replace the corresponding rules. The changes are restricted on the meta level.

**Definition 1** *Let $atom$ be a datatype that represents an atom, and $atoms$ be a datatype that represents a list of atoms in First-Order Logic (FOL). The functor $f/2/atom, atoms/atoms$ generates lists to concatenate individual atom. The set of meta predicates $\mathcal{M}_{meta}$ contains the following preserved special predicates to construct a meta interpreter. The predicate $solve/1/[atoms] \in \mathcal{M}_{meta}$ is a meta predicate to construct meta-level atoms, and the predicate $clause/2/[atom, atoms] \in \mathcal{M}_{meta}$ is a meta predicates to construct clauses.*

Algorithm. 1 shows a pseudocode of the DLMI workflow. The input is a logic program. **(Line 1)** Functor $f_{func}$ converts the input logic program ground atoms $\mathcal{G}_{\mathcal{P}}$ to lists $f_{func}(\mathcal{G}_{\mathcal{P}})$. **(Line 2)** Meta converter converts the input logic program ground atoms $\mathcal{G}_{\mathcal{P}}$ to lists $f(\mathcal{G}_{\mathcal{P}})$ using the meta predicates $solve$ and $clause$. **(Line 3)** Meta converter calculates and assigns weights $\mathcal{W}_{convert}$ to the meta ground atoms $\mathcal{G}_{meta}$. **(Line 4)** Meta interpreter takes the weighted meta ground atoms $\mathcal{W}_{convert} * \mathcal{G}_{meta}$ and meta rules as inputs and performs the differentiable forward reasoning (Evans & Grefenstette, 2018; Shindo et al., 2021b;a). Weights $W_{meta}$ of the outputs atoms are calculated in this step.

Our meta program for naive interpreter consists of two rules:

```
solve((A,B)):-solve(A),solve(B).
solve(A):-clause(A,B),solve(B).
```

The first meta rule defines conjunction, it can be extended to any number of arities with conjunction. Considering generating a list using the first rule, we can call the first meta rule recursively to generate a list. For example, if we use the first rule three times recursively on $A$, we will get $f(A, f(A, f(A, A)))$ as the answer. Using the first rule, the meta interpreter can generate a list that represents the body $B$ of a clause. The goal is defined by the second rule. The predicate $clause(A, B)$ checks whether the rule $A :- B$ exists in the algorithm that we want to evaluate. The predicate $solve(B)$ checks whether the body $B$ of the rule $A :- B$ is true.

To enable the naive interpreter to have an additional functionality of providing the proof tree, we construct the meta interpreter with minor modifications as:

```
solve((A,B),(proofA,proofB)):-solve(A,proofA),solve(B,proofB).
solve(A,(A:-proofB)):-clause(A,B),solve(B,proofB).
```

The predicate $clause$ keeps the same as the vanilla meta interpreter. The predicate $solve$ takes two entries as inputs, the first entry is an atom and the second entry is the corresponding proof of the atom. The first meta rule defines conjunction, not only for the atoms, but also for the proofs.

**Algorithm 1 DLMI Workflow**

```
Input: logic program $P$
1: functor $f_{func}$ converts input logic program ground atoms $\mathcal{G_P}$ to lists $f_{func}(\mathcal{G_P})$
2: meta converter generates meta ground atoms $\mathcal{G}_{meta}$ by wrapping up the atoms $\mathcal{G_P}$ and lists $f(\mathcal{G_P})$ using meta predicates $solve$ and $clause$
3: meta converter calculates and assigns weights $W_{convert}$ to meta ground atoms $\mathcal{G}_{meta}$
4: calculate output weights $W_{meta} = f_{infer}(W_{convert} * \mathcal{G}_{meta})$
5: return $W_{meta}$
```

| NSFR | DLMI |
|---|---|
| 1 kp(X):-in(01,X),in(02,X), in(03,X), in(04,X), same_shape_pair(01,02), same_color_pair(01,02), same_shape_pair(03,04),diff_color_pair(03,04). | 1 solve((A,B)):-solve(A),solve(B).<br>2 solve(A):-clause(A,B),solve(B). |
| 2 same_shape_pair(X,Y):-shape(X,Z),shape(Y,Z). | |
| 3 same_color_pair(X,Y):-color(X,Z),color(Y,Z). | |
| 4 diff_color_pair(X,Y):-color(X,Z),color(Y,W),diff_color(Z,W). | |

| Accuracy | |
|---|---|
| NSFR | DLMI |
| 100% | 100% |

Figure 4: **DLMI can realize Neuro-Symbolic Foward Reasoning (NSFR)**. The first comparison is the amount of codes used by two systems. The left side is the four rules used by NSFR and the right top is the two meta rules used by the meta interpreter. The second comparison in the right bottom shows the accuracy of the two systems achieved in the Kandinsky two pairs experiments.

## 4 Instantiations of Differentiable Meta-Logical Programming

We empirically demonstrate the following desired properties of DLMI on different tasks: (i) DLMI is an efficient instantiation of the differentiable forward reasoner, which is able to obtain the equivalent high quality results as the original (object level) reasoner while using less codes on the meta level. (ii) DLMI can reason about reasoning, i.e., DLMI realizes a differentiable reasoner with proof trees, which enables the system to trace back to provide the evidences for a certain decision. (iii) DLMI can reason about its operations (decisions), i.e., DLMI can interact with the given input under the given logical concepts, and provide evidences for doing such decision. All experiments were performed on an Intel i7-10700H 8-Core Processor with 16 GB RAM.

### 4.1 Differentiable Logical Reasoning

In this experiment, we show that our DLMI is able to obtain the equivalent high-quality results as the original reasoner while using much fewer codes on the meta level.

We adopt the experiment of classifying Kandinsky two pairs experiments shown in Shindo et al. (2021a); Holzinger et al. (2019; 2021). We test the Kandinsky plot on two systems, one is the original reasoner NSFR and the other one is NSFR within the DLMI framework (a similar structure as shown in the right part of fig. 2). The input of both systems is a Kandinsky two pairs plot, and the outputs of both systems are the atoms describing the shape and the color of these four objects. We compare the output atoms to evaluate the accuracy of DLMI and NSFR.

The Kandinsky patterns we evaluate are shown in the leftmost column of fig. 9. We use two criteria to identify whether a plot is Kandinsky two pairs: 1. The Kandinsky plot should have two pairs of objects in the same shape, 2. one pair of objects should be of the same color and the other pair should have a different color. We use four clauses to formalize the criteria into logic representation.

Fig. 4 shows a comparison of the rules and the accuracy of the output between the original reasoner NSFR and DLMI. DLMI uses much fewer codes on the meta level without sacrificing the high accuracy. Due to the limit of space, we show part of the outputs of both systems in fig. 5. The value in front of each atom shows the corresponding weight. Besides the label of the plot, DLMI is able to offer more information about the original algorithm. For example, the atom $clause(same\_shape\_pair(obj0, obj1), (shape(obj0, triangle), shape(obj0, triangle))$ has a weight of $0.98$. We can infer from this atom, that the rule:

| NSFR | DLMI |
|---|---|
| 0.98: color(obj0,red), 0.98:shape(obj0,triangle) | 0.99:solve(color(obj0,red)), 0.99:solve(shape(obj0,triangle)) |
| 0.98: color(obj1,blue), 0.98:shape(obj1,square) | 0.99:solve(color(obj1,blue)), 0.99:solve(shape(obj1,square)) |
| 0.98:diff_color_pair(obj0,obj1) | 0.99:solve(diff_color_pair(obj0,obj1)) |

Figure 5: Partial output atoms of NSFR and DLMI. The output atoms which have the same meaning have almost the same weight.

| NSFR | DLMI |
|---|---|
| N/A | 0.98: solve(same_shape_pair(obj0,obj2),proofs(same_shape_pair(obj0,obj2), proof((shape(obj0,triangle),0.98),(shape(obj2,triangle),0.98)))),<br>0.02: solve(same_shape_pair(obj0,obj1),proofs(same_shape_pair(obj0,obj1), proof((shape(obj0,triangle),0.98),(shape(obj1,triangle),0.05)))),<br>0.99: solve(shape(obj0,triangle),(shape(obj0,triangle),0.98)),<br>0.99: solve(shape(obj2,triangle),(shape(obj2,triangle),0.98)),<br>0.02: solve(shape(obj1,triangle),(shape(obj1,triangle),0.05)), |

Figure 6: Partial output examples of the DLMI with proof tree. Due to the initial lack of the ability of reasoning on the meta level, NSFR is unable to generate atoms with proof tree.

```
same_shape_pair(obj0,obj1):- shape(obj0,triangle),shape(obj0,triangle).
```

exists in the original algorithm.

## 4.2 Explainable Logical Reasoning

Our DLMI outperforms many other systems by having the ability to realize additional abilities on the meta level without touching the original system. This subsection illustrates the introspection ability of the meta interpreter by providing a proof tree. We use the Kandinsky two pairs experiment in this section as well.

We compare DLMI with NSFR in this experiment. NSFR is unable to generate atoms with proof tree due to the initial lack of the meta level reasoning. Fig. 6 shows part of the output results. We take three output atoms as examples to discuss in detail. The first example we would like to show is the atom $solve(shape(obj0, triangle), (shape(obj0, triangle), 0.98))$ with a weight of $0.99$. From looking solely at the proof part, the first information we can acquire is that this atom is a ground atom, since the proof of this atom (the latter part) has a single number $0.98$. The second information we can obtain, that this ground atom is true because the proof of the atom shows that the atom has a weight of value $0.98$.

As the second example, we discuss the atom

```
solve(same_shape_pair(obj0,obj2),
(same_shape_pair(obj0,obj2),
((shape(obj0,triangle),0.98), (shape(obj2,triangle),0.98)))).
```

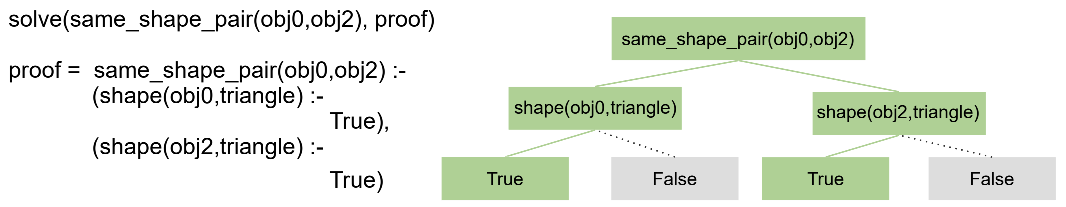
This atom has a weight of $0.98$, which indicates this atom is true. This atom consists of two parts, the first entry is the atom and the second entry is its corresponding proof. We rewrite the atom proof as shown in fig. 7 (left). The proof explains why this atom is true. From the proof, we can conclude that the atom $same\_shape\_pair(obj0, obj1)$ is deduced based on the rule:

```
same_shape_pair(obj0,obj1):-shape(obj0,triangle),shape(obj1,triangle).
```

Since both ground atoms $shape(obj0, triangle)$ and $shape(obj1, triangle)$ are true, the atom $same\_shape\_pair(obj0, obj1)$ is true.

The last example we would like to discuss is an atom with a small weight value. We take the atom with a weight value $0.02$:

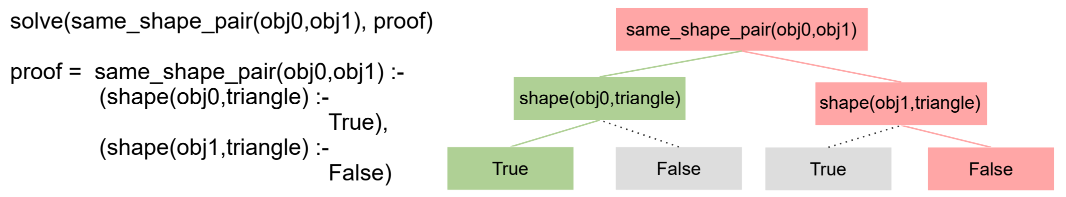
```
solve(same_shape_pair(obj0,obj1),
```



```
solve(same_shape_pair(obj0,obj2), proof)

proof =  same_shape_pair(obj0,obj2) :-
           (shape(obj0,triangle) :-
                                 True),
           (shape(obj2,triangle) :-
                                 True)
```

Figure 7: An example of the DLMI outputs with large weight. The left part of the plot shows an atom, it has two entries, the first entry is a logic atom and the second entry is its proof. The proof has a tree structure and can be rewritten into a hierarchical representation. The right side of the plot is a visualization of the corresponding atom proof tree structure. The color green indicates that the content in the box is $True$. We mark the color of the leaves or nodes which have no influence on its parent generation as gray.



```
solve(same_shape_pair(obj0,obj1), proof)

proof =  same_shape_pair(obj0,obj1) :-
           (shape(obj0,triangle) :-
                                 True),
           (shape(obj1,triangle) :-
                                 False)
```

Figure 8: An example of the DLMI outputs with small weight. The left part of the plot shows an atom, it has two entries, the first entry is the logic atom and the second entry is its proof. The proof has a tree structure and can be rewritten into a hierarchical representation. The right side of the plot is a visualization of the corresponding atom proof tree structure. The color green indicates that the content in the box is $True$, while the color red indicates that the content in the box is $False$. We mark the color of leaves or nodes which have no influence on its parent generation as gray.

```
(same_shape_pair(obj0,obj1),
((shape(obj0,triangle),0.98), (shape(obj1,triangle),0.02))))
```

as an example. Following the same technique, we rewrite the proof of the atom as shown in the left part of fig. 8. Besides obtaining the information that this atom is false, we can trace back by following the tree structure to determine which part of the statement is the cause. In this example, since the atom $same\_shape\_pair(obj0, obj2)$ is false, we trace back and check the atoms $shape(obj0, triangle)$ and $shape(obj1, triangle)$. Then we know it is the atom $shape(obj1, triangle)$ that leads $same\_shape\_pair(obj0, obj2)$ to have a small weight.

## 4.3 Differentiable Planning via Meta-Logical Programming

In this subsection, we propose a novel experiment "repairing Kandinsky patterns", the idea is to edit the objects in the Kandinsky two pairs pattern so that it agrees with a given logical concept. We take the Kandinsky two pairs patterns (the leftmost column of fig. 9) as our test figures. We set the goal positions of the four objects on a horizontal line, on a vertical line and on a diagonal line.

We consider constructing a differentiable planner within the meta interpreter framework to move the objects. By viewing planning as differentiable meta-logical programming, our system is able to perceive from a picture, reason about the picture, and even interact with the picture. To the best of the authors' knowledge, this is the first system equipped with all three of the introduced abilities.

We consider modifying the differentiable logical reasoner structure proposed by Shindo et al. (2021a) to construct a differentiable logical planner. As preparation, We define new predicates $move\_h$, $move\_v$, $step\_h$ and $step\_v$ for the planner.

**Definition 2** *The set of predicates $\mathcal{M}_{move}$ contains the following preserved special predicates to construct a logical planner. The predicate $move\_h/2/[object, direction] \in \mathcal{M}_{move}$ is a predicate to construct ground atoms for horizontal direction left and right. The predicate*

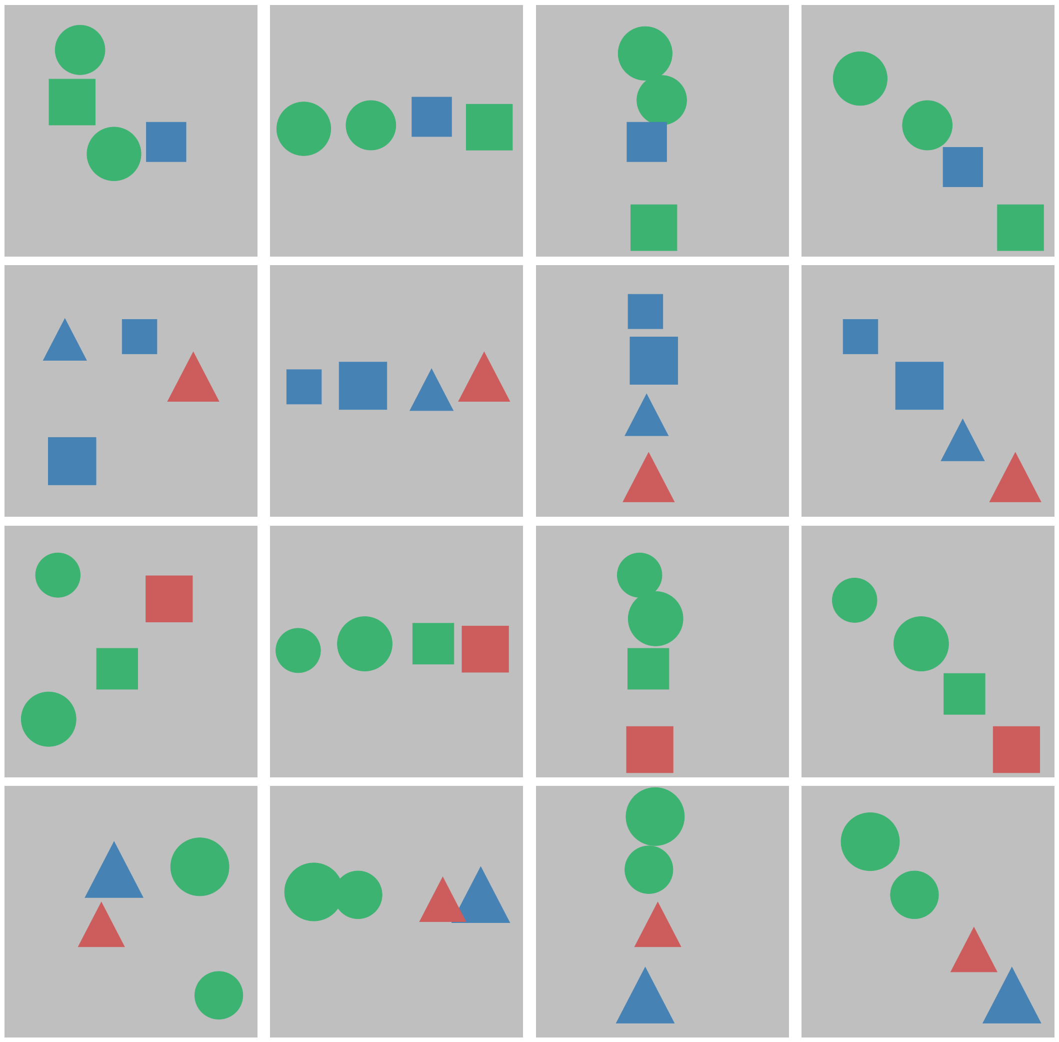


Figure 9: The meta-logical planner performed three separate experiments. The plots show the results of the experiments. The very left column plots show the original Kandinsky two-pairs plot, and the other three columns show the results of plots after planning step. From the left second column to the very right column, we set the goal positions of different objects on a horizontal, vertical, and diagonal line. We kindly refer to the appendix A for more "repairing Kandinsky patterns" plots.

*$move\_v/2/[object, direction] \in \mathcal{M}_{move}$ is a predicate to generate ground atoms for vertical direction up and down. $step\_h/2/[object, direction] \in \mathcal{M}_{move}$ is a predicate to construct ground atoms for horizontal steps, and $step\_v/2/[object, direction] \in \mathcal{M}_{move}$ is a predicate to construct ground atoms for vertical steps.*

To concatenate atoms generated by $move\_h$, $move\_v$, $step\_h$ and $step\_v$ for each object, we define a new predicate $move$. The atoms with predicate name $move$ is deduced by the rule:

```
move (X,A,B,C,D):-move_h(X,A),move_v(X,B),step_h(X,C),step_v(X,D).
```

In this rule, $X$ denotes the object, $A$ and $B$ denote the horizontal and vertical direction the object $X$ should move while $C$ and $D$ denote the number of steps that the object $X$ should move in the horizontal and vertical direction. We first discretize the picture into a $10 * 10$ picture and set the origin point at the center of the left bottom. We obtain the locations of different objects from the perception module (we use the predefined YOLO (Redmon et al., 2016) as the perception module. For details of the neural network parameters, we kindly refer to the article (Shindo et al., 2021a))

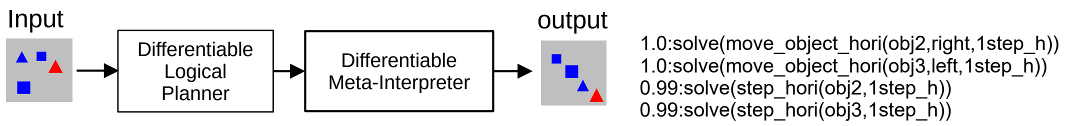

Input
Differentiable Logical Planner
Differentiable Meta-Interpreter
output
1.0:solve(move_object_hori(obj2,right,1step_h))
1.0:solve(move_object_hori(obj3,left,1step_h))
0.99:solve(step_hori(obj2,1step_h))
0.99:solve(step_hori(obj3,1step_h))

Figure 10: The left part of this plot shows the structure of the modified system. The right part of the plot shows part of the output atoms of the meta interpreter.

and use them as known values to calculate the direction and steps an object should move to reach the goal. For example, we may set the goal position of object 1 to $(x_g, y_g)$ and then get the current location $(x^l, y^l)$ of a object 1 from the perception module. We use the value of $(x^l - x_g)$ to determine the direction of the object's horizontal move. If the value is positive, then the object should move left. Otherwise, it should move right. Following the same procedure, we determine the vertical move direction for the object 1. As for steps, we use the absolute value of $(x_g - x^l)$ together with the discretized pixel length $d$ to acquire the step size in the horizontal and vertical direction. Assume the discretized pixel length is $d$, the horizontal length is $\left|x_1 - x^1\right|$, then the steps $s$ of the object 1 in the horizontal direction would be $s = \lfloor \left|x_1 - x^1\right| / l \rfloor$. Using the same technique, we can get the steps for moving the object in the vertical direction. After getting the direction and steps, we use the predicate $move\_h$, $move\_v$, $step\_h$ and $step\_v$ to generate the ground atoms for the planner and then the evaluation module would be called to assign weights for each atom, at the end, we perform the differentiable $move$ rule to concatenate the atoms for different objects.

Fig. 9 shows the three experiments performed by the planner on the "repairing Kandinsky patterns" task. The leftmost column of the plots show the original Kandinsky two pairs patterns and the other three columns show the plots after the planning step. From the second left column to the right most column, we set the goal positions of the objects on a horizontal line, on a vertical line and on a diagonal line. To realize the planner within the meta interpreter framework, we concatenate the meta interpreter with the planner. Fig. 10 (left) shows the structure of the modified system. We wrap the ground atoms of the planner with meta predicates $solve$ and $clause$. Then, following the same procedure of assigning weights for the atoms and preforming the differentiable forward inference, the results of the meta interpreter are shown in fig. 10 (right). Please note that, due to the page limit, the results shown in the figure are one small fraction of the complete output results. In practice, to decrease the memory usage in the meta interpreter, we split the move into two steps, and we modify the rules for each step. In the first step, we move the object horizontally and calculate the number of steps needed to move the object to the desired point. The corresponding rule for the first step is:

```
move (X,A,C):-move_h(X,A),step_h(X,C).
```

And the rule for the second step is:

```
move (X,B,D):-move_v(X,B),step_v(X,D).
```

$X$ stands for object, $A, B$ indicate the direction that the object should move towards, and $C, D$ indicate the steps in the corresponding direction.

## 5 Conclusions and Future Work

We proposed the first differentiable framework for meta-logical programming: Differentiable Logical-Meta Interpreter (DLMI), which can reason and learn logic programs. The DLMI framework is evaluated with three instantiations. Besides providing increased code efficiency without sacrificing accuracy, DLMI demonstrates the ability to integrate additional functionality (such as providing a proof tree) without interfering with the original system. Furthermore, by viewing planning as differentiable meta-logical programming, DLMI is able to easily carry out perception, reasoning, and interaction jointly with the input subject under a given logical concept.

Our differentiable meta-logical programming framework provides several interesting avenues for future work. One direction would be to define a loss function and use the meta interpreter to accomplish algorithmic supervision of a neural network. Other interesting directions include defining a loss function based on the proof tree to force the neural network to follow the right reasoning steps, and combining the meta interpreter with a Bayesian causal neural network.

## REFERENCES

Krzysztof R. Apt and Franco Turini. *Meta-Logics and Logic Programming*. MIT Press (MA), 1995.

Stefania Costantini. Meta-reasoning: A survey. In *Computational Logic: Logic Programming and Beyond*, 2002.

Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *J. Artif. Intell. Res.*, 61:1–64, 2018.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70, pp. 1126–1135, 2017.

Thomas L. Griffiths, Frederick Callaway, Michael B. Chang, Erin Grant, Paul M. Krueger, and Falk Lieder. Doing more with less: Meta-reasoning and meta-learning in humans and machines. *Current Opinion in Behavioral Sciences*, 29:24–30, 2019a.

Thomas L. Griffiths, Frederick Callaway, Michael B. Chang, Erin Grant, Paul M. Krueger, and Falk Lieder. Doing more with less: Meta-reasoning and meta-learning in humans and machines. *Current Opinion in Behavioral Sciences*, 29:24–30, 2019b.

P M Hill and J Gallagher. Meta-Programming in Logic Programming. In *Handbook of Logic in Artificial Intelligence and Logic Programming: Volume 5: Logic Programming*. Oxford University Press, 1998.

Andreas Holzinger, Michael Kickmeier-Rust, and Heimo Müller. Kandinsky patterns as iq-test for machine learning. In *Proceedings of the 3rd International Cross-Domain Conference for Machine Learning and Knowledge Extraction (CD-MAKE)*, pp. 1–14, 2019.

Andreas Holzinger, Anna Saranti, and Heimo Müller. Kandinsky patterns - an experimental exploration environment for pattern analysis and machine intelligence. *CoRR*, abs/2103.00519, 2021.

Timothy M. Hospedales, Antreas Antoniou, Paul Micaelli, and Amos J. Storkey. Meta-learning in neural networks: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 44(9):5149–5169, 2022.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross B. Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1988–1997, 2017.

Junkyung Kim, Matthew Ricci, and Thomas Serre. Not-so-clevr: learning same–different relations strains feedforward neural networks. *Interface focus*, 2018.

John W. Lloyd. *Foundations of Logic Programming, 1st Edition*. Springer, 1984.

Pattie Maes and D. Nardi. *Meta-Level Architectures and Reflection*. Elsevier Science Inc., USA, 1988.

Heimo Müller and Andreas Holzinger. Kandinsky patterns. *Artificial Intelligence*, 300:103546, 2021.

Alberto Pettorossi (ed.). *Proceedings of the 3rd International Workshop of Meta-Programming in Logic, (META-92)*, volume 649 of *Lecture Notes in Computer Science*, 1992. Springer.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents, 2022.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. A generalist agent, 2022.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Hoboken, New Jersey, 3rd edition, 2009.

Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

Hikaru Shindo, Devendra Singh Dhami, and Kristian Kersting. Neuro-symbolic forward reasoning. *CoRR*, abs/2110.09383, 2021a.

Hikaru Shindo, Masaaki Nishino, and Akihiro Yamamoto. Differentiable inductive logic programming for structured examples. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, pp. 5034–5041, 2021b.

Wolfgang Stammer, Patrick Schramowski, and Kristian Kersting. Right for the right concept: Revising neuro-symbolic concepts by interacting with their explanations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3619–3629, 2021.

Leon Sterling and Ehud Y Shapiro. *The art of Prolog: advanced programming techniques*. MIT press, 1994.

Sebastian Thrun and Lorien Pratt. *Learning to Learn: Introduction and Overview*, pp. 3–17. Springer US, Boston, MA, 1998.

## A Appendix A

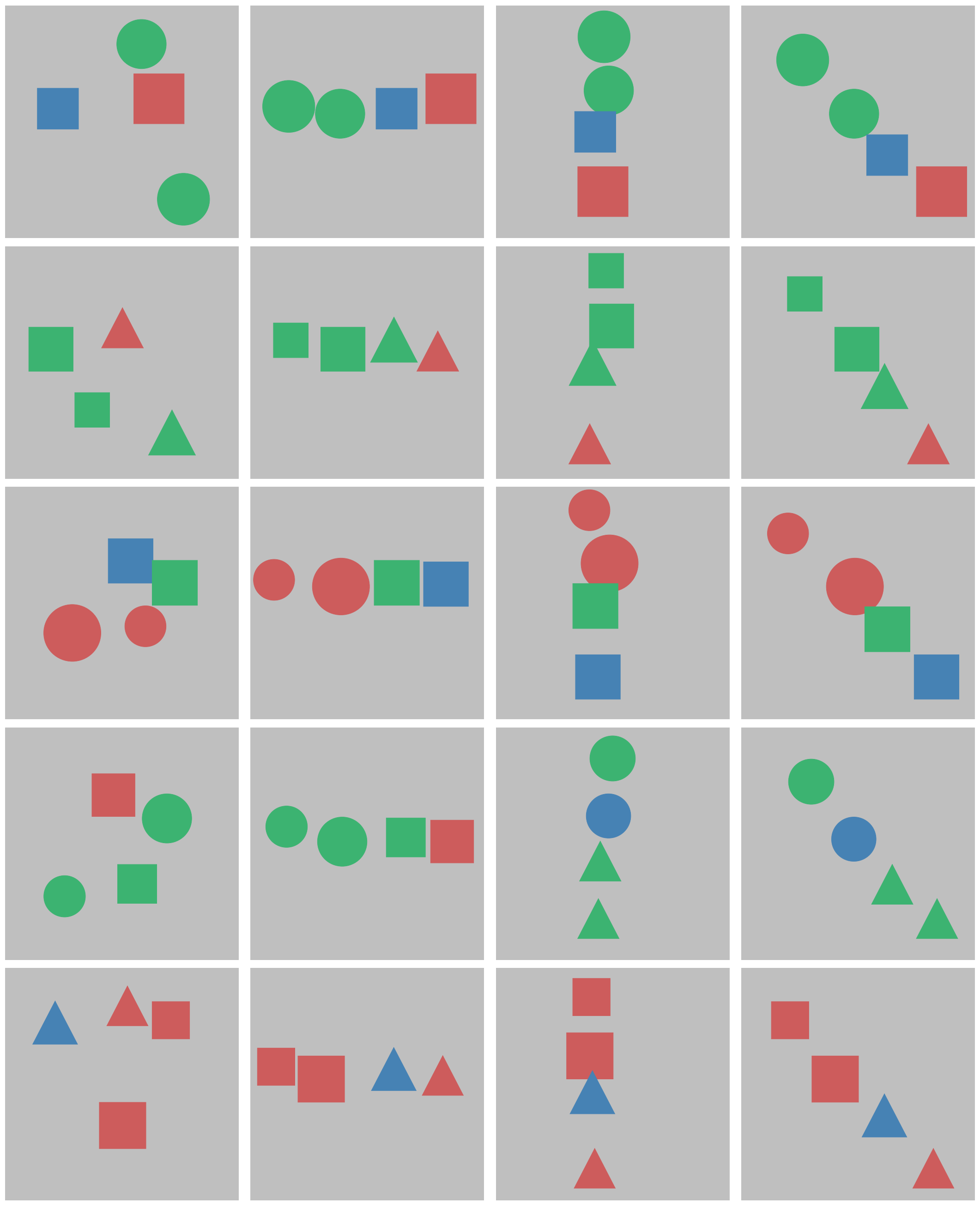


Figure 11: The meta-logical planner performed three separate experiments on different Kandinsky patterns. This plot shows the results of those experiments. The very left column plots show the original Kandinsky two-pairs plot, and the other three columns show the results of plots after planning step. From the left second column to the very right column, we set the goal positions of different objects on a horizontal, vertical, and diagonal line. We notice there may be some overlaps between objects. The reason lies in the box size we choose to discretize the plot, the smaller box size the plot get discretized, the better result we will get.

## B Appendix B

In this section, we introduce the comparisons between our baseline method NSFR and other methods. Table 1 shows the comparison in 2D Kandinsky classification experiments. Table 2 shows the comparison in 3D CLEVER-Hans experiments. For experiment settings and parameters please refer Shindo et al. (2021a) for details.

| | Training Data | | | Test Data | | |
|---|---|---|---|---|---|---|
| | NSFR | ResNet50 | YOLO+MLP | NSFR | ResNet50 | YOLO+MLP |
| Twopairs | **100.0** | **100.0** | **100.0** | **100.0** | 50.81 | 98.07 |
| Threepairs | **100.0** | **100.0** | **100.0** | **100.0** | 51.65 | 91.27 |
| Closeby | **100.0** | **100.0** | **100.0** | **100.0** | 54.53 | 91.40 |
| Red-Triangle | 95.80 | **100.0** | **100.0** | **95.60** | 57.19 | 78.37 |
| Online/Pair | 99.70 | **100.0** | **100.0** | **100.0** | 51.86 | 66.19 |
| 9-Circles | 96.40 | **100.0** | **100.0** | **95.20** | 50.76 | 50.76 |

Table 1: The classification accuracy in each data set. NSFR outperforms the considered baselines. Neural networks over-fit while training and perform poorly with testing data. Best results are bold.

| Model | Validation | Test | Validation | Test |
|---|---|---|---|---|
| | CLEVR-Hans3 | | CLEVR-Hans7 | |
| CNN | 99.55 | 70.34 | 96.09 | 84.50 |
| NeSy (Default) | 98.55 | 81.71 | 96.88 | 90.97 |
| NeSy-XIL | 100.00 | 91.31 | 98.76 | **94.96** |
| NS-FR | 98.18 | **98.40** | 93.60 | 92.19 |

Table 2: Classification accuracy of NSFR for CLEVR-Hans data sets compared to baselines.