Data-Efficient Learning with Neural Programs

Alaia Solko-Breslin, Seewon Choi, Ziyang Li, Neelay Velingker,
Rajeev Alur, Mayur Naik, Eric Wong
University of Pennsylvania
{alaia,seewon,liby99,neelay,alur,mhnaik,exwong}@seas.upenn.edu

Abstract

Many computational tasks can be naturally expressed as a composition of a DNN followed by a program written in a traditional programming language or an API call to an LLM. We call such composites "neural programs" and focus on the problem of learning the DNN parameters when the training data consist of end-to-end input-output labels for the composite. When the program is written in a differentiable logic programming language, techniques from neurosymbolic learning are applicable, but in general, the learning for neural programs requires estimating the gradients of black-box components. We present an algorithm for learning neural programs, called ISED, that only relies on input-output samples of black-box components. For evaluation, we introduce new benchmarks that involve calls to modern LLMs such as GPT-4 and also consider benchmarks from the neurosymbolic learning literature. Our evaluation shows that for the latter benchmarks, ISED has comparable performance to state-of-the-art neurosymbolic frameworks. For the former, we use adaptations of prior work on gradient approximations of black-box components as a baseline, and show that ISED achieves comparable accuracy but in a more data- and sample-efficient manner. [1]

1 Introduction

Many computational tasks cannot be solved by neural perception alone but can be naturally expressed as a composition of a neural model M_θ followed by a program P written in a traditional programming language or an API call to a large language model (LLM). We call such composites "neural programs" and study the problem of learning neural programs in an end-to-end manner with a focus on data and sample efficiency. One problem that is naturally expressed as a neural program is scene recognition [29], where M_θ classifies objects in an image and P prompts GPT-4 to identify the room type given these objects (Fig. 1).

Neurosymbolic learning [2] is one instance of neural program learning in which P takes the form of a logic program. DeepProbLog (DPL) [14] and Scallop [13] are frameworks that extend ProbLog and Datalog, respectively, to ensure that the symbolic component P is differentiable. This differentiability requirement is what facilitates learning in many neurosymbolic learning frameworks. There are also abductive learning frameworks that do not explicitly differentiate programs. Instead, they require that the symbolic component expose a method for abducing the function's inputs for a given output, often using Prolog for the symbolic component as a result [6, 23]. While logic programming languages are expressive enough for these frameworks to solve tasks such as sorting [14], visual question answering [13], and path planning [23], they offer restricted features and a narrow range of libraries, making them incompatible with calls to arbitrary APIs or to modern LLMs.

Learning neural programs when P is not expressed as a logic program is a difficult problem because gradients across black-box programs cannot be computed explicitly. One possible solution is to use REINFORCE [26] to sample symbols from distributions predicted by M_θ and compute the expected

[1] Code is available at https://github.com/alaiasolkobreslin/ISED

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

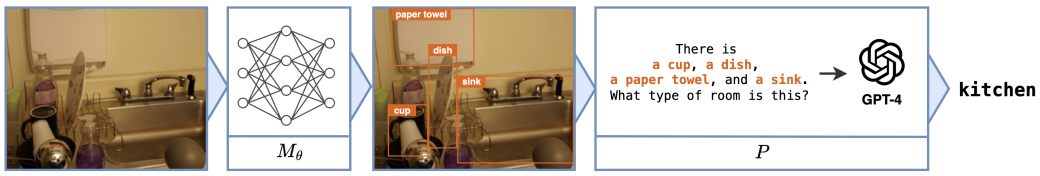

Figure 1: Neural program decomposition for scene recognition.

reward using the output label. However, REINFORCE is not sample-efficient as it produces a weak learning signal, especially when applied to programs with a large number of inputs. There are other REINFORCE-based methods that can be applied to the neural program learning setting, namely IndeCateR [21] and Neural Attention for Symbolic Reasoning (NASR) [5]. However, IndeCateR struggles with sample efficiency despite providing lower variance than REINFORCE, and NASR performs poorly when intermediate labels are unavailable for pretraining. Another possible solution is Approximate Neurosymbolic Inference (A-NeSI) [24], which trains a neural network to estimate the gradient of P, but learning the surrogate neural network becomes more difficult as the complexity of P increases. Moreover, the additional neural models in the learning framework in A-NeSI results in data inefficiency.

In this paper, we propose an algorithm for learning neural programs, based on reinforcement learning, which is compatible with arbitrary programs. Our approach, called ISED (Infer-Sample-Estimate-Descend), yields a framework that expands the applicability of neural program learning frameworks by providing a data- and sample-efficient method of training neural models with randomly initialized weights. ISED uses outputs of M_θ as a probability distribution over inputs of P and samples representative symbols u from this distribution. ISED then computes outputs v of P corresponding to these symbols. The resulting symbol-output pairs can be viewed as a symbolic program consisting of clauses of the form `if symbol` $= u$ `then output` $= v$ summarizing P. The final step is to estimate the gradient across this symbolic summary, inspired by ideas from the neurosymbolic learning literature, to propagate loss across the composite model.

Our evaluation considers 16 neural program benchmark tasks. Our results show that ISED outperforms purely neural networks and CLIP [19] on neural program tasks involving GPT-4 calls. Additionally, ISED outperforms neurosymbolic methods on 9 of the 14 benchmarks tasks that can be encoded in logic programming languages. ISED is also the top performer on 8 out of the 16 benchmark tasks when compared to REINFORCE-based and black-box gradient estimation baselines. Furthermore, we show that ISED is more data- and sample-efficient than baseline methods.

In summary, the main contributions of this paper are as follows: 1) we introduce neural programs as a generalization of neurosymbolic programs, 2) we introduce new tasks involving neural programs that use Python and calls to GPT-4 called neuroPython and neuroGPT programs, respectively, 3) we present ISED, a general algorithm for data- and sample-efficient learning with neural programs, and 4) we conduct a thorough evaluation using existing techniques against a diverse set of benchmarks.

2 Neural Programs

Problem Statement. In the neural program learning setting, we attempt to optimize model parameters M_θ which are being supervised by a fixed program P. Specifically, we are given a training dataset $\mathcal{D}$ of length N containing input-output pairs, i.e., $\mathcal{D} = \{(x_1, y_1), \ldots (x_N, y_N)\}$. Each x_i represents unstructured data (e.g., image data) whose corresponding structured data (intermediate labels) are not given. Each y_i is the result of applying P to the structured data corresponding to x_i. Given a loss function $\mathcal{L}$, we want to minimize the loss of $\mathcal{L}(P(M_\theta(x_i)), y_i)$ for each (x_i, y_i) pair in order to optimize θ. Loss minimization is straightforward when there is some mechanism for automatically differentiating programs, but we focus on the setting of optimizing θ without assuming the differentiability of P. We now introduce three motivating applications that can be framed in this

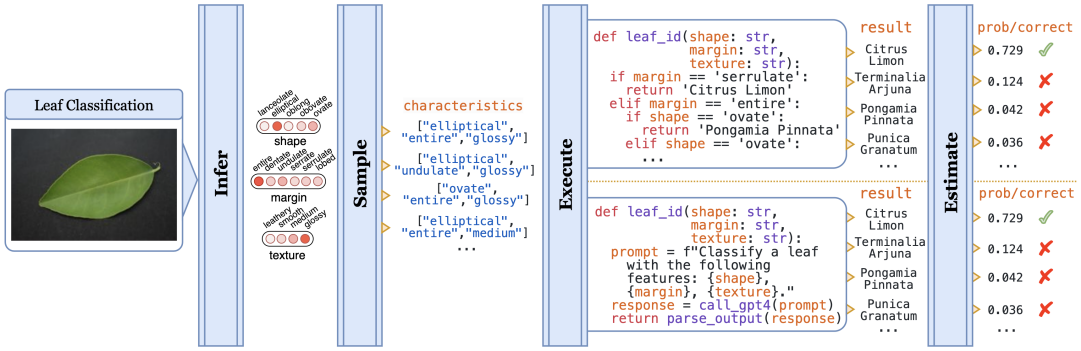

Figure 2: Illustration of our inference pipeline for the leaf classification task. `leaf_id` can be written with a decision tree (top program) or with a call to GPT-4 (bottom program).

setting, namely classifying images of leaves [9], scene recognition [29], and hand-written formula evaluation (HWF) [12].

Leaf Classification. We consider a real-world example that deals with the problem of classifying leaf images. Traditional neural methods predict the species directly, without explicit notion of leaf features such as margin, shape, and texture, resulting in solutions that are data-inefficient, inaccurate, and harder to understand.

We instead present a neural programming solution, making use of leaf classification decision trees [22]. These decision trees allow identifying plant species based on the visible characteristics of their leaves. Here, the neural model takes a leaf image and predicts its shape, margin, and texture. The program can then be written in two ways: one implementation involves encoding the decision tree in Python; another involves constructing a prompt using the predicted leaf features and calling GPT-4 (see Fig. 2). The latter is possible because ISED allows the use of black-box programs, so programs can also use state-of-the-art foundation models such as GPT-4 for computation.

Scene Recognition. The goal of this task is to classify images according to their room types. The model receives an image from a scene dataset [16] and predicts among the 9 different room types: bedroom, bathroom, dining room, living room, kitchen, lab, office, house lobby, and basement.

The traditional neural solution trains a convolutional neural network that directly predicts the room type. On the other hand, the neural program solution decomposes the task into detecting objects in the scene and identifying the room type based on those objects. We use an off-the-shelf object detection model YOLOv8 [20] and finetune it with a custom convolutional neural network to output labels related to scene recognition. We then make a GPT-4 call to predict the most likely room type given the list of detected objects.

Hand-written Formula. In this task, a model is given a list of hand-written symbols containing digits (0-9) and operators ($+$, $-$, $\times$, and $\div$) [12]. The dataset contains length 1-7 formulas free of syntax or divide-by-zero errors. The model is trained with supervision on the evaluated floating-point result without the label of each symbol. Since inputs are combinatorial and results are rational numbers, end-to-end neural methods struggle with accuracy. Meanwhile, neurosymbolic methods for this task either use specialized algorithms [12] or handcrafted differentiable programs [13].

With ISED, the program can be written in just a few lines of Python. It takes in a list of characters representing symbols, and simply invokes the Python `eval` function on the joined expression string. The `hwf` evaluation function can be used just like any other PyTorch [17] module since ISED internally performs sampling and probability estimation to estimate the gradient.

3 Learning Neural Programs

In this section, we present the intuition behind ISED and the values it approximates. Next, we introduce the programming interface for ISED, which lays the groundwork for presenting the algorithm. We then formally describe the steps of ISED.

3.1 ISED Overview

Assuming P is a black-box, we can collect symbol-output samples (u, v) from P. Such collection of samples can be viewed as a *summary* logic program consisting of rules of the form `if` $r = u$ `then` $y = v$. For instance, in the task of adding two digits r_1 and r_2, one rule of the logic program would be $r_1 = 1 \wedge r_2 = 2 \rightarrow y = 3$. Techniques from neurosymbolic literature via exact or approximate weighted model counting (WMC) [10] can then be used for computing the gradient across such a summary of P. However, having the complete summary of all combinations of symbols is not feasible for a black-box P. ISED samples symbols from the probability distribution predicted by the neural network M_θ, evaluates P on each sample, and takes the gradient across this partial summary of P. This is a good approximation of the complete summary since it is likely to contain symbols with high probability, which contribute the most in exact computation.

This approach differs from REINFORCE in how it differentiates through this summary of P. REINFORCE rewards sampled symbols that resulted in the correct output through optimizing the log probability of each symbol, weighted by reward values. This weighted-sum style estimation provides a weaker learning signal compared to WMC used by ISED, making learning harder for REINFORCE as the number of inputs to P increases. See Appendix A for further details.

3.2 Preliminaries and Programming Interface

ISED allows programmers to write black-box programs that operate on diverse structured inputs and outputs. To allow such programs to interact with neural networks, we define an interface named *structural mapping*. This interface serves to 1) define the data-types of black-box programs' input and output, 2) marshall and un-marshall data between neural networks and logical black-box functions, and 3) define the loss. We define a *structural mapping* τ as either a discrete mapping (with Σ being the set of all possible elements), a floating point, a permutation mapping with n possible elements, a tuple of mappings, or a list of up to n elements. We define τ inductively as follows:

$$\tau ::= \textsc{Discrete}(\Sigma) \mid \textsc{Float} \mid \textsc{Permutation}_n \mid \textsc{Tuple}(\tau_1, \ldots, \tau_m) \mid \textsc{List}_n(\tau)$$

Using this, we may further define data-types such as $\textsc{Integer}_j^k = \textsc{Discrete}(\{j, \ldots, k\})$, $\textsc{Digit} = \textsc{Integer}_0^9$, and $\textsc{Bool} = \textsc{Discrete}(\{\text{true}, \text{false}\})$. These types give ISED the flexibility learn neural programs with diverse types of inputs and outputs, e.g., $\textsc{Permutation}_n$ input and output types for integer list sorting and $\textsc{List}_9(\textsc{List}_9(\textsc{Digit}))$ for sudoku solving.

We also define a *black-box program* P as a function $(\tau_1, \ldots, \tau_m) \rightarrow \tau_o$, where $\tau_1, \ldots, \tau_m$ are the input types and τ_o is the output type. For example, the structural input mapping for the hand-written formula task is $\textsc{List}_7(\textsc{Discrete}(\{0, \ldots, 9, +, -, \times, \div\}))$, and the structural output mapping is $\textsc{Float}$. The mappings suggest that the program takes a list of length up to 7 as input, where each element is a digit or an arithmetic operator, and returns a floating point number.

There are two interpretations of a structural mapping: the set interpretation SET(τ) represents a mapping with defined values, e.g., a digit with value 8; the tensor interpretation DIST(τ) represents a mapping where each value is associated with a probability distribution, e.g., a digit that is 1 with probability 0.6 and 7 with probability 0.4. We use the set interpretation to represent structured program inputs that can be passed to a black-box program and the tensor interpretation to represent probability distributions for unstructured data and program outputs. These two interpretations are defined for the different structural mappings in Table 1.

Table 1: Set and tensor interpretations of different structural mappings.

Mapping (τ)	**Set Interpretation** (SET(τ))	**Tensor Interpretation** (DIST(τ))
$\textsc{Discrete}_\Sigma$	Σ	$\{\vec{v} \mid \vec{v} \in \mathbb{R}^{\lvert\Sigma\rvert}, v_i \in [0,1], i \in 1 \ldots \lvert\Sigma\rvert\}$
$\textsc{Float}$	$\mathbb{R}$	n/a
$\textsc{Permutation}_n$	$\{\rho \mid \rho \text{ is a permutation of } [1, \ldots, n]\}$	$\{[\vec{v_1}, \ldots, \vec{v_n}] \mid \vec{v_i} \in \mathbb{R}^n, v_{i,j} \in [0,1], i \in 1 \ldots n\}$
$\textsc{Tuple}(\tau_1, \ldots, \tau_m)$	$\{(a_1, \ldots, a_m) \mid a_i \in \text{SET}(\tau_i)\}$	$\{(a_1, .., a_m) \mid a_i \in \text{DIST}(\tau_i)\}$
$\textsc{List}_n(\tau')$	$\{[a_1, \ldots, a_j] \mid j \leq n, a_i \in \text{SET}(\tau')\}$	$\{[a_1, .., a_j] \mid j \leq n, a_i \in \text{DIST}(\tau')\}$

In order to represent the ground truth output as a distribution to be used in the loss computation, there needs to be a mechanism for transforming SET(τ) mappings into DIST(τ) mappings. For

this purpose, we define a *vectorize* function $\delta_\tau : (\text{SET}(\tau), 2^\tau) \rightarrow \text{DIST}(\tau)$ for the different output mappings τ in Table 2. When considering a datapoint (x, y) during training, ISED samples many symbols and obtains a list of outputs $\hat{y}$. The vectorizer then takes the ground truth y and the outputs $\hat{y}$ as input and returns the equivalent distribution interpretation of y. While $\hat{y}$ is not used by δ_τ in most cases, we include it as an argument so that FLOAT output mappings can be discretized, which is necessary for vectorization. For example, if the inputs to the vectorizer for the hand-written formula task are $y = 2.0$ and $\hat{y} = [1.0, 3.5, 2.0, 8.0]$, then it would return $[0, 0, 1, 0]$.

Table 2: Vectorize and aggregate functions of different structural mappings.

Mapping (τ)	**Vectorizer** ($\delta_\tau(y, \hat{y})$)	**Aggregator** ($\sigma_\tau(\hat{r}, \hat{p})$)
DISCRETE_n	$e^{(y)}$ with dim n	$\hat{p}[\hat{r}]$
FLOAT	$[\mathbf{1}_{y=\hat{y}_i}$ for $i \in [1, \ldots, \text{length}(\hat{y})]]$	n/a
PERMUTATION_n	$[\delta_{\text{DISCRETE}_n}(y[i])$ for $i \in [1, \ldots, n]]$	$\otimes_{i=1}^{n} \sigma_{\text{DISCRETE}_n}(\hat{r}[i], \hat{p}[i])$
$\text{TUPLE}(\tau_1, \ldots, \tau_m)$	$[\delta_{\tau_i}(y[i])$ for $i \in [1, \ldots, m]]$	$\otimes_{i=1}^{m} \sigma_{\tau_i}(\hat{r}[i], \hat{p}[i])$
$\text{LIST}_n(\tau')$	$[\delta_{\tau'}(a_i)$ for $a_i \in y]$	$\otimes_{i=1}^{n} \sigma_{\tau'}(\hat{r}[i], \hat{p}[i])$

We also require a mechanism to aggregate the probabilities of sampled symbols that resulted in a particular output. With this aim, we define an *aggregate* function $\sigma_\tau : (\text{SET}(\tau), \text{DIST}(\tau)) \rightarrow \mathbb{R}$ for different input mappings τ in Table 2. ISED aggregates probabilities either by taking their minimum or their product, and we denote both operations by $\otimes$. The aggregator takes as input sampled symbols $\hat{r}$ and neural predictions $\hat{p}$ from which $\hat{r}$ was sampled. It gathers values in $\hat{p}$ at each index in $\hat{r}$ and returns the result of $\otimes$ applied to these values. For example, suppose we use `min` as the aggregator $\otimes$ for the hand-written formula task. Then if $\otimes$ takes $\hat{r} = [1, +, 1]$ and $\hat{p}$ as inputs where $\hat{p}[0][1] = 0.1$, $\hat{p}[1][+] = 0.05$, and $\hat{p}[2][1] = 0.1$, it would return 0.05.

3.3 Algorithm

We now formally present the ISED algorithm. For a given task, there is a black-box program P, taking m inputs, that operates on structured data. Let $\tau_1, ..., \tau_m$ be the mappings for these inputs and τ_o the mapping for the program's output. We write P as a function from its input mappings to its output mapping: $P : (\tau_1, ..., \tau_m) \rightarrow \tau_o$. For each unstructured input i to the program, there is a neural model $M_{\theta_i}^{i} : x_i \rightarrow \text{DIST}(\tau_i)$. S is a sampling strategy (e.g., categorical sampling) that samples symbols using the outputs of a neural model, and k is the number of samples to take for each training example. There is also a loss function $\mathcal{L}$ whose first and second arguments are the predicted and ground truth values respectively. We present the pseudocode of the algorithm in Algorithm 1 and describe its steps with the hand-written formula task:

Infer. The training pipeline starts with an example from the dataset, (x, y) = ([ ,  , ], 3.0), and uses a CNN to predict these images, as shown on lines 3-4. ISED initializes $\hat{p} = M_\theta(x)$.

Sample. ISED samples $\hat{r}$ from $\hat{p}$ for k iterations using sampling strategy S. For each sample j, the algorithm initializes $\hat{r}_j$ to be the sampled symbols, as shown on lines 6-9. To continue our example, suppose ISED initializes $\hat{r}_j = [7, +, 2]$ for sample j. The next step is to execute the program on $\hat{r}_j$, as shown on line 10, which in this example means setting $\hat{y}_j = P(\hat{r}_j) = 9.0$.

Estimate. In order to compute the prediction value to use in the loss function, ISED must consider each output y_l in the output mapping and accumulate the aggregated probabilities for all sampled symbols that resulted in the output y_l. We specify $\otimes$ as the `min` function, and $\oplus$ as the `max` function in this example. Note that ISED requires that $\otimes$ and $\oplus$ represent either `min` and `max` or `mult` and `add` respectively. We refer to these two options as the `min-max` and `add-mult` semirings. We define an *accumulate* function ω that takes as input an element of the output mapping y_l, sampled outputs $\hat{y}$, sampled symbols $\hat{r}$, and predicted input distributions $\hat{p}$. The accumulator performs the $\oplus$ operation on aggregated probabilities for elements of $\hat{y}$ that are equal to y_l and is defined as follows:

$$\omega(y_l, \hat{y}, \hat{r}, \hat{p}) = \oplus_{j=1}^{k} \mathbf{1}_{\hat{y}_j = y_l} \sigma_{\tau_o}(\hat{r}_j, \hat{p}_j)$$

Continuing our example, suppose, among the samples, there are two symbolic combinations ($[7, +, 2]$ and $[3, *, 3]$) that resulted in the output 9.0. Let us say that these sets of symbols had probabilities $[0.3, 0.8, 0.8]$ and $[0.1, 0.1, 0.1]$, respectively. Then the result of the probability aggregation for $y_l = 9.0$ would be $\omega(9.0, \hat{y}, \hat{r}, \hat{p}) = \texttt{max}(\texttt{min}([0.3, 0.8, 0.8]), \texttt{min}([0.1, 0.1, 0.1])) = 0.3$.

Algorithm 1 ISED training pipeline

Require: P is the black-box program $(\tau_1, \ldots, \tau_m) \to \tau_o$, $M^i_{\theta_i}$ the neural model $x_i \to \text{DIST}(\tau_i)$ for each τ_i, S the sampling strategy, k the sample count, $\mathcal{L}$ the loss function, and $\mathcal{D}$ the dataset.

1: **procedure** TRAIN
2: **for** $((x_1, \ldots x_m), y) \in \mathcal{D}$ **do**
3: **for** $i \in 1 \ldots m$ **do**
4: $\hat{p}[i] \leftarrow M^i_{\theta_i}(x_i)$ ▷ **Infer**
5: **end for**
6: **for** $j \in 1 \ldots k$ **do**
7: **for** $i \in 1 \ldots m$ **do**
8: Sample $\hat{r}_j[i]$ from $\hat{p}[i]$ using S ▷ **Sample**
9: **end for**
10: $\hat{y}_j \leftarrow P(\hat{r}_j)$
11: **end for**
12: $\hat{w} \leftarrow \text{normalize}([\omega(y_l, \hat{y}, \hat{r}, \hat{p}) \text{ for } y_l \in \tau_o \text{ (or } y_l \in \hat{y})])$ ▷ **Estimate**
13: $w \leftarrow \delta(y, \hat{y})$
14: $l \leftarrow \mathcal{L}(\hat{w}, w)$
15: Compute $\frac{\partial l}{\partial \theta}$ by performing back-propagation on l
16: Optimize θ based on $\frac{\partial l}{\partial \theta}$ ▷ **Descend**
17: **end for**
18: **end procedure**

ISED then sets $\tilde{w} = [\omega(y_l, \hat{y}, \hat{r}, \hat{p}) \text{ for } y_l \in \tau_o]$ in the case where τ_o is not FLOAT. When τ_o is FLOAT, as for hand-written formula, it only considers $y_l \in \hat{y}$. Next, it performs L_2 normalization over each element in $\tilde{w}$ and sets $\hat{w}$ to this result. To initialize the ground truth vector, it sets $w = \delta(y, \hat{y})$. ISED then initializes $l = \mathcal{L}(\hat{w}, w)$ and computes $\frac{\partial l}{\partial \theta_i}$ for each input i. These steps are shown on lines 12-15. In our running example, since 9.0 is an incorrect output, the probability of the first symbol being equal to 7 (instead of the correct answer 1) will be penalized while the probabilities for predicting other symbols are unchanged.

Descend. The last step is shown on line 16, where the algorithm optimizes θ_i for each input i based on $\frac{\partial l}{\partial \theta_i}$ using a stochastic optimizer (e.g., Adam optimizer). This completes the training pipeline for one example, and the algorithm returns all final θ_i after iterating through the entire dataset.

4 Evaluation

In this section, we evaluate ISED and aim to answer the following research questions:

RQ1: How does ISED compare to state-of-the-art neurosymbolic, REINFORCE-based, and gradient estimation baselines in terms of accuracy?
RQ2: What is the sample efficiency of ISED when compared to REINFORCE-based algorithms?
RQ3: How data-efficient is ISED compared to neural gradient estimation methods?

4.1 Benchmark Tasks: NeuroGPT, NeuroPython, and Neurosymbolic

We first introduce two new neural program learning benchmarks which both contain a program component that can make a call to GPT-4. We call such models neuroGPT programs.

Leaf Classification. In this task, we use a dataset, which we call LEAF-ID, containing leaf images of 11 different plant species [4], containing 330 training samples and 110 testing samples. We define custom DISCRETE types MARGIN, SHAPE, TEXTURE. With this, we define LEAF-TRAITS = TUPLE(MARGIN, SHAPE, TEXTURE) and LEAF-OUTPUT to be the DISCRETE set of 11 plant species in the dataset. Neural program solutions either prompt GPT-4 (GPT leaf) or use a decision tree (DT leaf).

Scene Recognition. We use a dataset containing scene images from 9 different room types [16], consisting of 830 training examples and 92 testing examples. We define custom types OBJECTS and SCENES to be DISCRETE set of 45 objects and 9 room types, respectively. We freeze the parameters

Table 3: Performance on selected benchmarks. "TO" means time-out, and "N/A" means the task could not be programmed in the framework. Methods are divided (from top to bottom) by neurosymbolic, black-box gradient estimation, and REINFORCE-based.

	Accuracy (%)							
Method	sum_2	sum_3	sum_4	HWF	DT leaf	GPT leaf	scene	sudoku
DPL	95.14	93.80	TO	TO	39.70	N/A	N/A	TO
Scallop	91.18	91.86	80.10	96.65	81.13	N/A	N/A	TO
A-NeSI	**96.66**	94.39	78.10	3.13	78.82	72.40	61.46	26.36
REINFORCE	74.46	19.40	13.84	88.27	40.24	53.84	12.17	79.08
IndeCateR	96.48	93.76	92.58	95.08	78.71	69.16	12.72	66.50
NASR	6.08	5.48	4.86	1.85	16.41	17.32	2.02	**82.78**
ISED (ours)	80.34	**95.10**	**94.10**	**97.34**	**82.32**	**79.95**	**68.59**	80.32

of YOLOv8 and only optimize the custom neural network. The neural program solution prompts GPT-4 to classify the scene.

We also consider several tasks from the neurosymbolic literature, including hand-written formula (HWF) evaluation and Sudoku solving. While the solutions to many of these tasks are usually presented as a logic program in neurosymbolic learning frameworks, neural program solutions can take the form of Python programs. We call such models neuroPython programs.

MNIST-R. MNIST-R [13, 14] contains 11 tasks operating on inputs of images of handwritten digits from the MNIST dataset [11]. This synthetic test suite includes tasks performing arithmetic (sum_2, sum_3, sum_4, mult_2, mod_2, add-mod-3, add-sub), comparison (less-than, equal), counting (count-3-or-4), and negation (not-3-or-4) over the digits depicted in the images. Each task dataset has a training set of 5K samples and a testing set of 500 samples.

HWF. The goal of the HWF task is to classify images of handwritten digits and arithmetic operators and evaluate the formula [12]. The dataset contains 10K formulas of length 1-7, with 1K length 1 formulas, 1K length 3 formulas, 2K length 5 formulas, and 6K length 7 formulas.

Visual Sudoku. The goal of this task is to solve an incomplete 9x9 Sudoku, where the problem board is given as MNIST digits. We follow the experimental setting of NASR [5], including their pre-trained MNIST digit recognition models and sudoku solvers. We use the SatNet dataset consisting of 9K training samples and 500 test samples [25].

4.2 Evaluation Setup and Baselines

All of our experiments were conducted on a machine with two 20-core Intel Xeon CPUs, one NVIDIA RTX 2080 Ti GPU, and 755 GB RAM. Unless otherwise noted, the sample count, i.e., the number of calls to the program P per training example, is fixed at 100 for all relevant methods. For additional details on experimental setup, see Appendix B. We apply a timeout of 10 seconds per testing sample, and report the average accuracy and 1-sigma standard deviation obtained from 10 randomized runs.

We pick as baselines neurosymbolic methods DeepProbLog (DPL) [14] and Scallop [13], A-NeSI [24] which performs neural approximation of the gradients, and sampling-based gradient approximation methods REINFORCE [26], IndeCateR [21], and NASR [5]. IndeCateR achieves provably lower variance than REINFORCE by using a specialized sampling method (Appendix A), and NASR is a variant specialized for efficient finetuning by using a single sample and a custom reward function. We also use purely neural baselines and CLIP [19] for GPT leaf and scene. CLIP is a multimodal model that supports zero-shot image classification by simply providing names of the output categories.

4.3 RQ1: Performance and Accuracy

To answer **RQ1**, we evaluate ISED's accuracy against those of the baselines. ISED matches, and in many cases surpasses, the accuracy of neurosymbolic and gradient estimation baselines. We highlight the results for sum_n from MNIST-R and other benchmarks in Table 3. Tables 7-10 in Appendix C

Table 4: Performance comparisons for sum_8, sum_{12}, and sum_{16} with different sample counts k.

	Accuracy (%)					
	sum_8		sum_{12}		sum_{16}	
Method	$k = 80$	$k = 800$	$k = 120$	$k = 1200$	$k = 160$	$k = 1600$
REINFORCE	8.32	8.28	7.52	8.20	5.12	6.28
IndeCateR	5.36	**89.60**	4.60	77.88	1.24	5.16
IndeCateR+	10.20	88.60	6.84	**86.92**	4.24	**83.52**
ISED (Ours)	**87.28**	87.72	**85.72**	86.72	**6.48**	8.13

contain results for the remaining MNIST-R tasks, including standard deviations for all tasks. ISED is the top performer on 8 out of the 16 total tasks.

On the GPT leaf and scene tasks, ISED outperforms the purely neural baseline by 3.82% and 31.42% respectively, and zero-shot CLIP by 59.80% and 17.50%. For many tasks, A-NeSI is the non-neurosymbolic method that comes closest to ISED, sometimes outperforming our method. However, A-NeSI achieves significantly lower performance than ISED on tasks involving complex programs, namely HWF and sudoku. This is likely due to the difficulty of training a neural model to estimate the output of P and its gradient when P is complex. ISED also outperforms REINFORCE on all but 3 tasks due to the REINFORCE learning signal being weaker for tasks where P involves multiple inputs. NASR outperforms ISED only on sudoku by 2.46% due to NASR being well-suited for fine-tuning as it restricts its algorithm to use a single sample. IndeCateR achieves similar performance compared to ISED on most tasks but achieves significantly lower accuracy on the scene classification task, which has a large input space with maximum 10 objects each with 47 possible values in each scene, demonstrating that IndeCateR is less sample-efficient than ISED. We elaborate more on this point in **RQ2**.

ISED outperforms the neurosymbolic methods on 9 out of 14 tasks that can be written in logic programming languages. Despite treating P as a black-box, ISED even outperforms Scallop on HWF by 0.69% and comes within 1.16% of NGS, a specialized neurosymbolic learning framework that uses abductive reasoning [12]. Furthermore, DPL timed out on 4 tasks, and Scallop timed out on 1 (sudoku). These results demonstrate that even for tasks that can be written in a logic programming language, treating the program as a black-box can often yield optimal results.

4.4 RQ2: Sample Efficiency

To answer **RQ2**, we evaluate the sample efficiency of ISED against REINFORCE, IndeCateR, and IndeCateR+ on adding MNIST digits. IndeCateR+ [21] is a variant of IndeCateR with a sampling method and loss computation customized for higher dimensional setting such as the addition of 16 MNIST digits. We vary the size of the input and output space (sum_8, sum_{12}, sum_{16}) of P as well as the sample count, and report the average accuracy and standard deviation obtained from 5 randomized runs (Tables 4, 11-13).

For a lower number of samples, ISED outperforms all other methods on the three tasks, outperforming IndeCateR by over 80% on sum_8 and sum_{12}. The experimental findings support the conceptual difference of REINFORCE-based methods providing a weak learning signal compared to ISED (Section 3.1). While ISED achieves accuracy similar to the top performer for sum_8 and sum_{12} with a high sample count, it comes second on sum_{16} with IndeCateR+ beating ISED by 75.39%. This suggests our approach is limited in scaling to high-dimensional inputs to P, and motivates exploring better sampling techniques, which is the core difference between IndeCateR and IndeCateR+.

4.5 RQ3: Data Efficiency

We now examine how ISED compares to state-of-the-art baselines in terms of data efficiency. We compare ISED and A-NeSI in terms of training time and accuracy on sum_3 and sum_4. We choose these tasks for evaluation because A-NeSI has been shown to scale well to multi-digit addition tasks [24]. Furthermore, these tasks come from the MNIST-R suite in which we use 5K training samples, which is less than what A-NeSI would have used in its evaluation (20K training samples for sum_3

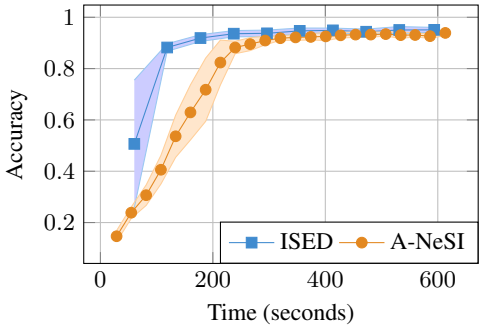

Figure 3: Accuracy vs. Time for sum_3.

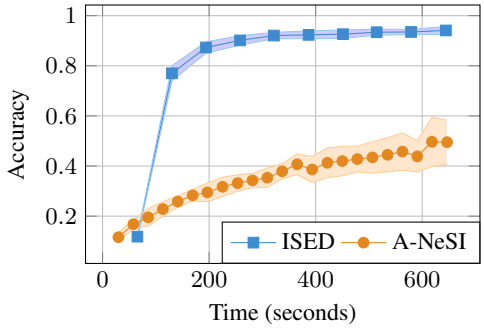

Figure 4: Accuracy vs. Time for sum_4.

and 15K for sum_4). We plot the average test accuracy and standard deviation vs. training time (over 10 runs) in Figures 3 and 4, where each point represents the result of 1 epoch.

While ISED and A-NeSI learn at about the same rate for sum_3 after about 5 minutes of training, ISED learns at a much faster rate for the first 5 minutes, reaching an accuracy of 88.22% after just 2 epochs (Fig. 3). The difference between ISED and A-NeSI is more pronounced for sum_4, with ISED reaching an accuracy of 94.10% after just 10 epochs while A-NeSI reaches 49.51% accuracy at the end of its 23rd epoch (Fig. 4). These results demonstrate that with limited training data, ISED is able to learn more quickly than A-NeSI, even for simple tasks. This result is likely due to A-NeSI training 2 additional neural models in its learning pipeline compared to ISED, with A-NeSI training a prior as well as a model to estimate the program output and gradient.

5 Limitations and Future Work

The main limitation of ISED is the difficulty of scaling with the dimensionality of the space of inputs to the program P. There are interesting future directions in adapting and expanding ISED for high dimensionality. Specifically, improvements to the sampling strategy could help adapt ISED to a complex space of inputs. Techniques can be borrowed from the field of Bayesian optimization where such large spaces have traditionally been studied. Furthermore, there is merit to systematically combining white-box and black-box methods. ISED is especially useful when logic programs fail to encode reasoning components. Therefore, we believe that ISED can be used as an underlying engine for a new neurosymbolic language that blends the accessibility of black-box with the performance of white-box methods.

6 Related Work

Neurosymbolic programming frameworks. These frameworks provide a general mechanism to define white-box neurosymbolic programs. DeepProbLog [14] and Scallop [13] abstract away gradient calculations behind a rule-based language. Others specialize in targeted applications, such as NeurASP [28] for answer set programming, or NeuralLog [3] for phrase alignment in NLP. ISED is similar in that it seeks to make classes of neurosymbolic programs easier to write and access; however, it diverges by offering an interface not bound by any specific domain or language syntax.

RL and sampling-based neurosymbolic frameworks. ISED incorporates concepts found in the RL algorithm REINFORCE [26] such as the sampling of actions according to the current policy distribution, similar to NASR [5], and IndeCateR [21]. Other work has proposed a semantic loss function for neurosymbolic learning which measures how well neural network outputs match a given constraint [27]. While this technique resembles ISED in that it samples symbols from their predicted distributions to derive the loss, it relies on symbolic knowledge in the form of a constraint in Boolean logic, whereas ISED allows the program component to be any black-box program.

Specialized neurosymbolic methods. The majority of the neurosymbolic learning literature pertains to point solutions for specific use cases [7, 25]. In the HWF example, NGS [12] and several of its variants leverage a hand-defined syntax defining the inherent structure within mathematical expressions. Similarly, DiffSort [18] leverages the symbolic properties of sorting to produce differentiable

sorting networks. Other point solutions address broader problem setups, such as NS-CL [15] which provides a framework for visual question answering by learning symbolic representations in text and images. For reading comprehension, the NeRd [3] framework converts NL questions into executable programs over symbolic information extracted from text. ISED aligns with all of these point solutions by aiming to solve problems that have thus far required technically specific solutions in order to access the advantages of neurosymbolic learning, but it takes an opposite and easier approach by forgoing significant specializations and instead leverages existing solutions as black-boxes.

Differentiable programming and non-differentiable optimization. Longstanding libraries in deep learning have grown to great popularity for their ability to abstract away automatic differentiation behind easy-to-use interfaces. PyTorch [17] is able to do so by keeping track of a dynamic computational graph. Similarly, JAX [1] leverages functional programming to abstract automatic differentiation. ISED follows the style of these frameworks by offering an interface to abstract away gradient calculations for algorithms used in deep learning, but ISED improves upon them by allowing systematic compatibility of non-differentiable functions.

7 Conclusion

We proposed ISED, a data- and sample-efficient algorithm for learning neural programs. Unlike existing general neurosymbolic frameworks which require differentiable logic programs, ISED is compatible with Python programs and API calls to GPT, and it employs a sampling-based technique to learn neural model parameters using forward evaluation. We showed that for neuroGPT, neuroPython, and neurosymbolic benchmarks, ISED achieves better accuracy than end-to-end neural models and similar accuracy compared to neurosymbolic frameworks. ISED also often achieves superior accuracy on complex programs compared to REINFORCE-based and gradient estimation baselines. Furthermore, ISED learns in a more data- and sample-efficient manner compared to these baselines.

8 Acknowledgements

We thank the anonymous reviewers for useful feedback. This research was supported by ARPA-H grant D24AC00253-00, NSF award CCF 2313010, and by a gift from AWS AI to ASSET (Penn Engineering Center on Trustworthy AI).

References

[1] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[2] Swarat Chaudhuri, Kevin Ellis, Oleksandr Polozov, Rishabh Singh, Armando Solar-Lezama, Yisong Yue, et al. Neurosymbolic programming. *Foundations and Trends in Programming Languages*, 7(3):158–243, 2021.

[3] Zeming Chen, Qiyue Gao, and Lawrence S. Moss. NeuralLog: Natural language inference with joint neural and logical reasoning. In *Proceedings of *SEM 2021: The Tenth Joint Conference on Lexical and Computational Semantics*, pages 78–88, 2021.

[4] Siddharth Singh Chouhan, Uday Pratap Singh, Ajay Kaul, and Sanjeev Jain. A data repository of leaf images: Practice towards plant conservation with plant pathology. In *2019 4th International Conference on Information Systems and Computer Networks (ISCON)*, pages 700–707, 2019.

[5] Cristina Cornelio, Jan Stuehmer, Shell Xu Hu, and Timothy Hospedales. Learning where and when to reason in neuro-symbolic inference. In *International Conference on Learning Representations*, 2023.

[6] Wang-Zhou Dai, Qiuling Xu, Yang Yu, and Zhi-Hua Zhou. Bridging machine learning and logical reasoning by abductive learning. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019.

[7] Rajdeep Dutta, Qincheng Wang, Ankur Singh, Dhruv Kumarjiguda, Li Xiaoli, and Senthilnath Jayavelu. S-reinforce: A neuro-symbolic policy gradient approach for interpretable reinforcement learning. *arXiv preprint arXiv:2305.07367*, 2023.

[8] Jiani Huang, Ziyang Li, Binghong Chen, Karan Samel, Mayur Naik, Le Song, and Xujie Si. Scallop: From probabilistic deductive databases to scalable differentiable reasoning. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, pages 25134–25145, 2021.

[9] Paul Shekonya Kanda, Kewen Xia, and Olanrewaju Hazzan Sanusi. A deep learning-based recognition technique for plant leaf classification. *IEEE Access*, 9:162590–162613, 2021.

[10] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. Algebraic model counting. *CoRR*, 2012.

[11] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[12] Qing Li, Siyuan Siyuan Huang, Yining Hong, Yixin Chen, Ying Nian Wu, and Song-Chun Zhu. Closed loop neural-symbolic learning via integrating neural perception, grammar parsing, and symbolic reasoning. In *Proceedings of the 37th International Conference on Machine Learning*, page 5884–5894, 2020.

[13] Ziyang Li, Jiani Huang, and Mayur Naik. Scallop: A language for neurosymbolic programming. In *ACM International Conference on Programming Language Design and Implementation*, page 1463–1487, 2023.

[14] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, page 3753–3763, 2018.

[15] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *International Conference on Learning Representations*, 2019.

[16] Lukas Murmann, Michael Gharbi, Miika Aittala, and Fredo Durand. A multi-illumination dataset of indoor object appearance. In *2019 IEEE International Conference on Computer Vision (ICCV)*, Oct 2019.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, page 8026–8037, 2019.

[18] Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Differentiable sorting networks for scalable sorting and ranking supervision. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[19] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

[20] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[21] Lennert De Smet, Emanuele Sansone, and Pedro Zuidberg Dos Martires. Differentiable sampling of categorical distributions using the catlog-derivative trick. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, 2023.

[22] Dagher R. Talhouk S.N., Fabian M. Landscape plant database, 2015.

[23] Efthymia Tsamoura, Timothy Hospedales, and Loizos Michael. Neural-symbolic integration: A compositional perspective. In *AAAI Conference on Artificial Intelligence*, 2020.

[24] Emile van Krieken, Thiviyan Thanapalasingam, Jakub M. Tomczak, Frank van Harmelen, and Annette ten Teije. A-nesi: A scalable approximate method for probabilistic neurosymbolic inference. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, 2023.

[25] Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *Proceedings of the 36th International Conference on Machine Learning*, pages 6545–6554, 2019.

[26] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992.

[27] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *Proceedings of the 35th International Conference on Machine Learning*, pages 5502–5511, 2018.

[28] Zhun Yang, Adam Ishay, and Joohyung Lee. Neurasp: Embracing neural networks into answer set programming. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 1755–1762, 2020.

[29] Delu Zeng, Minyu Liao, Mohammad Tavakolian, Yulan Guo, Bolei Zhou, Dewen Hu, Matti Pietikäinen, and Li Liu. Deep learning for scene classification: A survey, 2021.

A Explanation of Differences Between ISED and Baseline Methods

We explain the differences between ISED and prior techniques using the simple example of sum_2, where digits are restricted to be between 0-2. Suppose that we are training a neural network M_θ for this task, and we are considering the symbol-output sample where the ground truth symbols are 1 and 2, i.e., $r_1 = 1, r_2 = 2$, and the ground truth output is $y = 3$. Suppose that the predicted distributions from M_θ for r_1 and r_2 are $[0.1, 0.6, 0.3]$ and $[0.2, 0.1, 0.7]$ respectively. We now explain how different methods perform their loss computations.

A.1 ISED

Suppose ISED is initialized with a sample count of 3, and the sampled symbol-output pairs are $((1, 2), 3)$, $((1, 0), 1)$, and $((2, 1), 3)$. We use the `add-mult` semiring in this example. ISED can be thought of as differentiating through the following summary logic program:

$$r_1 = 1 \wedge r_2 = 2 \rightarrow y = 3$$
$$r_1 = 1 \wedge r_2 = 0 \rightarrow y = 1$$
$$r_1 = 2 \wedge r_2 = 1 \rightarrow y = 3$$

As a result, the final vector calculated for the loss function before normalization would be

$$\begin{bmatrix} 0.0 \\ 0.6 * 0.2 \\ 0.0 \\ 0.6 * 0.7 + 0.3 * 0.1 \\ 0.0 \end{bmatrix}$$

where each value corresponds to the probability of the given output (possible outputs are in the range 0-4). Note that if there are duplicate samples, ISED includes the duplicate probabilities in its aggregation. In our implementation, we would perform normalization on this vector and then pass it into the binary cross-entropy loss function, with the ground truth vector being:

$$\begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 1.0 \\ 0.0 \end{bmatrix}$$

We would then minimize this loss and update M_θ accordingly.

If we use the `min-max` semiring instead, $*$ is replaced by `min` and $+$ by `max` in the final vector calculation, resulting in

$$\begin{bmatrix} 0.0 \\ 0.2 \\ 0.0 \\ 0.6 \\ 0.0 \end{bmatrix}$$

A.2 REINFORCE

Suppose REINFORCE is also initialized with a sample count of 3, and it samples the same symbol-output pairs. The final reward is computed by element-wise multiplication of the log probability of each sample with its reward value and taking the mean, as follows:

$$\frac{1}{3} * \begin{bmatrix} \log(0.6) + \log(0.7) \\ \log(0.6) + \log(0.2) \\ \log(0.3) + \log(0.1) \end{bmatrix} * \begin{bmatrix} 1.0 \\ 0.0 \\ 1.0 \end{bmatrix}$$

and the goal is to optimize M_θ to maximize this reward. While this approach resembles ISED's loss computation for the `add-mult` semiring, it does not involve the `mult` step. As it rewards possible values instead of possible combinations, the final reward would have been the same when (1,1) and (2,2) were the correct samples, instead of (1,2) and (2,1). Hence, the learning signal is weaker compared to ISED when there is more than one input to P.

A.3 IndeCateR and NASR

IndeCateR is an extension of the REINFORCE estimator that is unbiased with a provably lower variance. It assumes and exploits the factoring of the underlying multivariate distribution into independent categorical variables by summing out one dimension while keeping a sample for other dimensions fixed. For each sample drawn, IndeCateR systematically creates additional samples that differ on a single entry by enumerating all possible values for each variable. NASR targets efficient finetuning by setting the sample count to one and customizing the reward function.

The loss computation for IndeCateR and NASR are identical to that of REINFORCE, also providing weak signals with fewer samples. Furthermore, both set the reward to 0 for samples leading to incorrect predictions, effectively ignoring them, unlike ISED which penalizes such symbols. Since only the correct symbol contribute to the final reward, the signal is sparser than ISED, making it sample-inefficient.

A.4 A-NeSI

A-NeSI trains two additional neural networks: a prediction model $Q_{\theta'}$ as a surrogate for P, and a prior model R_α for learning the parameters α for the Dirichlet distribution D_α.

Suppose A-NeSI is also initialized with a sample count of 3. At each training step, A-NeSI first updates α using $\hat{y} = M_\theta(x)$. Next, it samples a single symbol from each of the 3 distributions sampled from D_α, and uses the sampled symbol-output pair and the standard cross entropy loss to update $Q_{\theta'}$. Then, A-NeSI optimizes M_θ by minimizing the loss $\mathcal{L}(Q_{\theta'}(M_\theta(r_1, r_2)))$ using the prediction model instead of P.

A.5 DeepProbLog

DeepProbLog (DPL) enumerates all possible proofs for each output and aggregates probabilities accordingly. For example, the proofs for $y = 1$ include $r_1 = 0, r_2 = 1$ and $r_1 = 1, r_2 = 0$. Thus, the probability of this output is $0.6 * 0.2 + 0.1 * 0.1$. The final vector calculated would be

$$\begin{bmatrix} 0.1 * 0.2 \\ 0.6 * 0.2 + 0.1 * 0.1 \\ 0.1 * 0.7 + 0.6 * 0.1 + 0.3 * 0.2 \\ 0.6 * 0.7 + 0.3 * 0.1 \\ 0.3 * 0.7 \end{bmatrix}$$

and we would pass this vector into some loss function (e.g., cross-entropy), with the same ground truth vector that ISED would use. DPL would then minimize this loss and update M_θ accordingly.

A.6 Scallop

Suppose Scallop is configured to use the `diff-top-1-proofs` semiring. This means that for each possible output, Scallop will use the proof of that output with the highest probability. For instance, the most likely proof for $y = 1$ is $r_1 = 1$ and $r_2 = 0$, and the probability of the output $y = 1$ is $0.6 * 0.2$. The final vector calculated would be

$$\begin{bmatrix} 0.1 * 0.2 \\ 0.6 * 0.2 \\ 0.1 * 0.7 \\ 0.6 * 0.7 \\ 0.3 * 0.7 \end{bmatrix}$$

and we would pass this vector into some loss function (e.g., binary-cross-entropy), with the same ground truth vector that ISED would use. Scallop would then minimize this loss and update M_θ accordingly. This probability estimation would change depending on the choice of semiring (e.g., `diff-top-k-proofs` for a different value of k).

B Evaluation Setup

For tasks with the MNIST dataset as unstructured data, we employ LeNet [11], a 2-layer CNN-based model, except for sum_8, sum_{12}, and sum_{16} tasks where we choose a smaller 2-layer CNN used by

IndeCateR [21]. For HWF, we also use a 2-layer CNN-based model. For leaf classification tasks, images are scaled down and passed to a simple CNN-based network with 4 convolutional layers. For the scene recognition, we use YOLOv8 and a 3-layer convolutional network for neural program methods, 7-layer CNN for the purely neural solution, and CLIP with ViT-B/32. For all tasks included in **RQ1**, other than sudoku, we remove the final softmax function at the end of each network when evaluating IndeCateR since its sampling procedure yields optimal results without the softmax. We also do that same with REINFORCE if it results in higher accuracy. Since sudoku uses a pretrained CNN, we use the same CNN across all methods, including IndeCateR and REINFORCE.

We use the Adam optimizer with the best learning rate among {1e−3, 5e−4, 1e−4}. We train for maximum 100 epochs, but stop early if the training saturates. For MNIST-R tasks, we used learning rate 1e−4 and trained ISED for 10 epochs, REINFORCE and IndeCateR for 50 epochs, and A-NeSI and NASR for 100 epochs. We trained ISED for 30 epochs, A-NeSI for 100 epochs, and the rest for 50 epochs for HWF and Leaf Classification with learning rate 1e−4. For the Scene Recognition task, we trained A-NeSI and the purely neural baseline 50 epochs and the rest 100 epochs with learning rate 5e−4. For tasks sum_8 to sum_{16} we trained ISED for 50 epochs and the rest for 100 epochs with learning rate 1e−3. For Visual Sudoku, we follow the setting in NASR [5] and train for 10 epochs with learning rate 1e−5.

We configure ISED to use the `min-max` semiring for HWF and the `add-mult` semiring for all other tasks. We use categorical sampling and binary cross-entropy loss for ISED.

B.1 Neural-GPT Experiment Prompts

For leaf classification and scene recognition, the neural-GPT experiments, we used the up-to-date version of GPT-4, `gpt-4-1106-preview` and `gpt-4o` respectively, with the parameter `top-p` set to 1e−8. We present the prompts used for the experiments in Tables 5 and 6.

Table 5: GPT-4 prompt for the leaf classification task.

System message	You are an expert in classifying plant species based on the margin, shape, and texture of the leaves. You are designed to output a single JSON.
User message	<PLANT NAME>. Classify into one of: <MARGIN/SHAPE/TEXTURE>. Give your answer without explanation.

Table 6: GPT-4 prompt for the scene recognition task.

System message	You are an expert at identifying room types based on the object detected. Give short single responses.
User message	There are <DETECTED OBJECTS>. What type of room is most likely? Choose among <SCENES>.

We use MARGIN = {entire, dentate, lobed, serrate, serrulate, undulate}, SHAPE ={elliptical, lanceolate, obovate, oblong, ovate}, TEXTURE ={glossy, leathery, smooth, medium}, and PLANT NAME ∈{Alstonia Scholaris, Citrus limon, Jatropha curcas, Mangifera indica, Ocimum basilicum, Platanus orientalis, Pongamia Pinnata, Psidium guajava, Punica granatum, Syzygium cumini, Terminalia Arjuna}.

Furthermore, SCENES = {bathroom, bedroom, dining room, living room, kitchen, lab, office, home lobby, basement} and DETECTED OBJECTS is a list of maximum length 10 with duplicates.

C Full Performance Summary

We report the accuracy of all benchmarks with 1-sigma standard deviation in Tables 7, 8, 9, and 10. We further provide the performance comparison with varying sample counts with 1-sigma standard deviation in Tables 11, 12, and 13.

Table 7: Performance comparison for DT leaf, GPT leaf, scene, and sudoku.

	Accuracy (%)			
Method	DT leaf	GPT leaf	scene	sudoku
DPL	39.70 ± 6.55	N/A	N/A	TO
Scallop	81.13 ± 3.50	N/A	N/A	TO
A-NeSI	78.82 ± 4.42	72.40 ± 12.24	61.46 ± 14.18	26.36 ± 12.68
REINFORCE	40.24 ± 0.08	53.84 ± 0.04	12.17 ± 0.02	79.08 ± 0.87
IndeCateR	78.71 ± 5.59	69.16 ± 2.35	12.72 ± 2.51	66.50 ± 1.37
NASR	16.41 ± 1.79	17.32 ± 1.92	2.02 ± 0.23	$\mathbf{82.78 \pm 1.06}$
ISED (ours)	$\mathbf{82.32 \pm 4.15}$	$\mathbf{79.95 \pm 5.71}$	$\mathbf{68.59 \pm 1.95}$	80.32 ± 1.79

Table 8: Performance comparison for HWF, sum_2, sum_3, and sum_4.

	Accuracy (%)			
Method	HWF	sum_2	sum_3	sum_4
DPL	TO	95.14 ± 0.80	93.80 ± 0.54	TO
Scallop	96.65 ± 0.13	91.18 ± 13.43	91.86 ± 1.60	80.10 ± 20.4
A-NeSI	3.13 ± 0.72	$\mathbf{96.66 \pm 0.87}$	94.39 ± 0.77	78.10 ± 19.0
REINFORCE	88.27 ± 0.02	74.46 ± 26.29	19.40 ± 4.52	13.84 ± 2.26
IndeCateR	95.08 ± 0.41	96.48 ± 0.53	93.76 ± 0.47	92.58 ± 0.80
NASR	1.85 ± 0.27	6.08 ± 0.77	5.48 ± 0.77	4.86 ± 0.93
ISED (ours)	$\mathbf{97.34 \pm 0.26}$	80.34 ± 16.14	$\mathbf{95.10 \pm 0.95}$	$\mathbf{94.1 \pm 1.6}$

Table 9: Performance comparison for mult_2, mod_2, less-than, and add-mod-3.

	Accuracy (%)			
Method	mult_2	mod_2	less-than	add-mod-3
DPL	95.43 ± 0.97	96.34 ± 1.06	$\mathbf{96.60 \pm 1.02}$	95.28 ± 0.93
Scallop	87.26 ± 24.70	77.98 ± 37.68	80.02 ± 3.37	75.12 ± 21.64
A-NeSI	96.25 ± 0.76	96.89 ± 0.84	94.75 ± 0.98	77.44 ± 24.60
REINFORCE	$\mathbf{96.62 \pm 0.23}$	94.40 ± 2.81	78.92 ± 2.31	$\mathbf{95.42 \pm 0.37}$
IndeCateR	96.32 ± 0.50	$\mathbf{97.04 \pm 0.39}$	94.98 ± 1.50	78.52 ± 23.26
NASR	5.34 ± 0.68	20.02 ± 2.67	49.30 ± 2.14	33.38 ± 2.81
ISED (ours)	96.02 ± 1.13	96.68 ± 0.93	96.22 ± 0.95	83.76 ± 12.89

Table 10: Performance comparison for add-sub, equal, not-3-or-4, and count-3-4.

	Accuracy (%)			
Method	add-sub	equal	not-3-or-4	count-3-4
DPL	93.86 ± 0.87	$\mathbf{98.53 \pm 0.37}$	98.19 ± 0.55	TO
Scallop	92.02 ± 1.58	71.60 ± 2.29	97.42 ± 0.73	93.47 ± 0.83
A-NeSI	93.95 ± 0.60	77.89 ± 36.01	98.63 ± 0.50	93.73 ± 2.93
REINFORCE	17.86 ± 3.27	78.26 ± 3.96	$\mathbf{99.28 \pm 0.21}$	87.78 ± 1.14
IndeCateR	93.74 ± 0.44	98.18 ± 0.39	99.26 ± 0.16	94.30 ± 1.26
NASR	5.26 ± 1.10	81.72 ± 1.94	68.36 ± 1.54	25.26 ± 1.66
ISED (ours)	$\mathbf{95.32 \pm 0.81}$	96.02 ± 1.74	98.08 ± 0.72	$\mathbf{95.26 \pm 1.04}$

Table 11: Performance comparison for sum_8 with different sample counts k.

	Accuracy (%)	
	sum_8	
Method	$k = 80$	$k = 800$
REINFORCE	8.32 ± 2.52	8.28 ± 0.39
IndeCateR	5.36 ± 0.26	$\mathbf{89.60 \pm 0.98}$
IndeCateR+	10.20 ± 1.12	88.60 ± 1.09
ISED (Ours)	$\mathbf{87.28 \pm 0.76}$	87.72 ± 0.86

Table 12: Performance comparison for sum_{12} with different sample counts k.

	Accuracy (%)	
	sum_{12}	
Method	$k = 120$	$k = 1200$
REINFORCE	7.52 ± 1.92	8.20 ± 1.80
IndeCateR	4.60 ± 0.24	77.88 ± 6.68
IndeCateR+	6.84 ± 2.06	$\mathbf{86.92 \pm 1.36}$
ISED (Ours)	$\mathbf{85.72 \pm 2.15}$	86.72 ± 0.48

Table 13: Performance comparison for sum_{16} with different sample counts k.

	Accuracy (%)	
	sum_{16}	
Method	$k = 160$	$k = 1600$
REINFORCE	5.12 ± 1.91	6.28 ± 1.04
IndeCateR	1.24 ± 1.68	5.16 ± 0.52
IndeCateR+	4.24 ± 0.95	$\mathbf{83.52 \pm 1.75}$
ISED (Ours)	$\mathbf{6.48 \pm 0.50}$	8.13 ± 1.10

D License Information

For implementing the baselines, we adapted the code from the official repositories of DeepProblog [14] (Apache 2.0), Scallop [8] (MIT), A-NeSI [24] (MIT), NASR [5] (MIT), and IndeCateR [21] (Apache 2.0). Additionally, our benchmarks use Multi-illumination dataset [16] (CC BY 4.0), HWF dataset (CC BY-NC-SA 3.0) from NGS [12], a subset of the leaf database [4] (CC BY 4.0), YOLOv8 (AGPL-3.0) and CLIP (MIT).

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction claim that our proposed algorithm (ISED) achieves similar, and often superior accuracy, compared to state-of-the-art neurosymbolic learning, REINFORCE-based, and black-box gradient estimation frameworks, but in a more data- and sample-efficient manner. We provide evidence for these claims in our evaluation in Sections 4.3, 4.4, and 4.5.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We have a separate section on Limitations (Section 5), and show limitation in scaling to high dimensional inputs with experimental results in RQ2 (Section 4.4).

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: Our paper does not include any theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We release code for all baselines and ISED to reproduce the results reported in the paper. The hyperparameters used for experiments are summarized in Appendix B.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We are releasing data and code, along with the instructions for reproducing all results. We include directions for downloading datasets and setting up the environment.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide information on data splits in Section 4.1 and provide the rest of the experimental details in Appendix B.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: All results are accompanied by standard deviation with clearly stated factors of variability. While some tables are presented without standard deviations (Tables 3 and 4), 1-sigma standard deviation is provided for these results in Tables 7-13 in Appendix C.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide details of the compute used in Section 4.2. We also specify the number of runs and epochs for each task (Appendix B) and training time for selected benchmarks (Section 4.5).

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the Code of Ethics and and followed them whenever relevant.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: We propose a new framework for learning neural networks in presence of black-box components and there is no societal impact to be pointed out.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: We do not release new data or models, hence the work poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: For datasets, code, and pretrained models used, we cited the original paper and provided the license information in Appendix D. We stated the versions of GPT-4 and YOLO in Appendix B.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.

- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper releases code for the proposed learning algorithm, which is well documented with license and training information. The limitations are mentioned in the paper (Sections 4.4 and 5).

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This paper did not involve any crowdsourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper did not involve any crowdsourcing or research with human subjects.

    Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.