arXiv:2406.17224v1 [cs.AI] 25 Jun 2024

Large Language Models are Interpretable Learners

Ruochen Wang[*]
UCLA

Si Si
Google Research

Felix Yu
Google Research

Dorothea Wiesmann
Google Research

Cho-Jui Hsieh
Google Research

Inderjit Dhillon
Google Research

https://github.com/ruocwang/llm-symbolic-program

Abstract

The trade-off between expressiveness and interpretability remains a core challenge when building human-centric predictive models for classification and decision-making. While symbolic rules offer interpretability, they often lack expressiveness, whereas neural networks excel in performance but are known for being black boxes. In this paper, we show a combination of Large Language Models (LLMs) and symbolic programs can bridge this gap. In the proposed LLM-based Symbolic Programs (LSPs), the pretrained LLM with natural language prompts provides a massive set of interpretable modules that can transform raw input into natural language concepts. Symbolic programs then integrate these modules into an interpretable decision rule. To train LSPs, we develop a divide-and-conquer approach to incrementally build the program from scratch, where the learning process of each step is guided by LLMs. To evaluate the effectiveness of LSPs in extracting interpretable and accurate knowledge from data, we introduce IL-Bench, a collection of diverse tasks, including both synthetic and real-world scenarios across different modalities. Empirical results demonstrate LSP's superior performance compared to traditional neurosymbolic programs and vanilla automatic prompt tuning methods. Moreover, as the knowledge learned by LSP is a combination of natural language descriptions and symbolic rules, it is easily transferable to humans (interpretable), and other LLMs, and generalizes well to out-of-distribution samples.

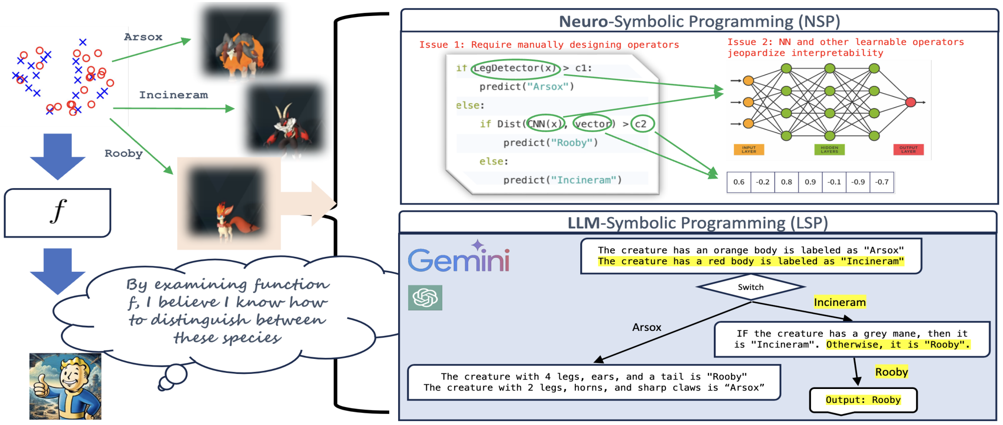
1 Introduction

Learning interpretable predictive models from annotated data remains a key challenge in human-centric AI. Given input-output pairs $\{(x_i, y_i)\}$, the objective is to learn a function $f : x \to y$ that not only fits the data accurately but is also interpretable. tIn this context, a strong form of "interpretable" means that individuals with no prior domain knowledge can understand and apply the decision rules demonstrated by f, facilitating *the transfer of knowledge from AI to humans*. This is crucial not only for enhancing the transparency of AI systems but also for enabling humans to learn from these models, empowering various human-in-the-loop applications such as scientific discovery, material synthesis, and automatic data annotation [Chaudhuri et al., 2021].

Consider an exemplar task of classifying species in Palworld [Pair, 2024] - a newly released Pokemon-style game - based on a few image-label pairs, as illustrated in Figure 1. The ultimate goal is that even humans unfamiliar with Palworld can replicate AI's decisions by following the same predictive rules after examining the model trained on the data. This task effectively represents the challenge of extracting interpretable knowledge, such as species characteristics, from data. The algorithm we propose in this paper learns a model following the decision rule illustrated in Figure 1, which is

[*] Work completed during internship at Google.

Preprint. Under review.


Arsox
Incineram
Rooby
f
By examining function f, I believe I know how to distinguish between these species
Neuro-Symbolic Programming (NSP)
Issue 1: Require manually designing operators
Issue 2: NN and other learnable operators jeopardize interpretability
INPUT LAYER
HIDDEN LAYERS
OUTPUT LAYER
0.6
-0.2
0.8
0.9
-0.1
-0.9
-0.7
LLM-Symbolic Programming (LSP)
Gemini
Switch
Arsox
Incineram
Rooby

```
if LegDetector(x) > c1:
    predict("Arsox")
else:
    if Dist(CNN(x), vector) > c2
        predict("Rooby")
    else:
        predict("Incineram")
```

The creature has an orange body is labeled as "Arsox"
The creature has a red body is labeled as "Incineram"

IF the creature has a grey mane, then it is "Incineram". Otherwise, it is "Rooby".

The creature with 4 legs, ears, and a tail is "Rooby"
The creature with 2 legs, horns, and sharp claws is "Arsox"

Output: Rooby

Figure 1: **Illustration of LLM-Symbolic vs. Neuro-Symbolic Program on interpretable learning task.** The goal is to develop a model that allows humans with no prior knowledge to replicate AI's decisions by following the same rules as the model. While **NSP (Top right)** offers a certain level of interpretability, it heavily relies on manually designing operators, and the inclusion of neural operators often reduces interpretability. In contrast, **LSP (Bottom right)** generates fully interpretable programs with the help of versatile LLM modules.

designed to be easily understood and reproduced by humans. In essence, this problem can be viewed as discovering interpretable knowledge (e.g., the properties of a species in Palworld) from the data.

Despite extensive research, the problem of developing a fully interpretable predictive model has not been fully addressed. Traditional methods often face a trade-off between expressiveness and interpretability: Deep neural networks, for instance, are powerful yet operate as "black boxes". Although post-hoc explanation methods attempt to make these models more transparent by identifying influential features [Ribeiro et al., 2016], they do not clarify the underlying decision-making processes and have no control over the learning process. Directly learning interpretable models like (locally) linear, tree-based often falls short in expressiveness, especially with complex inputs like images.

To address this challenge, **Neurosymbolic Programs (NSPs)** [Chaudhuri et al., 2021, Shah et al., 2020, Cui and Zhu, 2021, Nauta et al., 2021b] offer a promising solution by modeling the decision rule as a program incorporating both symbolic operations and neural network modules. Despite this, the inherent trade-off between expressiveness and interpretability persists. While the integration of neural modules enhances expressiveness, it also compromises the program's overall interpretability. Additionally, designing effective symbolic operators requires significant expertise and is critical for the performance of the resulting program, necessitating careful customization for each specific dataset [Chaudhuri et al., 2021, Shah et al., 2020, Cui and Zhu, 2021].

Is it possible to harness the power of neural networks within Neurosymbolic Programs without compromising interpretability? This paper presents an affirmative answer. Our key insight is that (Multimodal) LLMs encompass a variety of powerful, conditional probabilistic sub-models. These models share a unified parametric architecture with the unconditional parent LLM (Super Model), yet distinctive defined by their respective prompts. Therefore, crafting prompts (by either Human or meta-LLMs) for LLM is equivalent to searching over the hypothesis space spanned by these submodels. This yields an infinite set of neural network-based operations that are inherently interpretable and can serve as fundamental "learnable" building blocks within Neurosymbolic Programs.

Building on this insight, we introduce a novel framework termed **LLM-Symbolic Programs (LSPs)**, defined and learned through LLMs. Our approach leverages a minimal Domain-Specific Language (DSL) set with only two operators: prompted-LLM and conditional branching, yielding a classic decision-making process structured as trees. We then propose a learning algorithm to incrementally learn the tree using LLMs with prompt optimization. To thoroughly evaluate the efficacy of LSPs, we construct the **Interpretable-Learning-Benchmark** of diverse predictive tasks, containing both synthetic and real-world data across vision and text modalities. Our empirical findings show that LSPs surpass the accuracy of both traditional XAI methods and LLMs prompted with automatically learned instructions, all while maintaining human interpretability. These results highlight the potential of LSPs to significantly enhance the performance and utility of Multimodal LLMs in various applications.

2 LLM-Symbolic Programs

This section explains our proposed framework: LLM-Symbolic Programs. Section 2.1 reviews Neurosymbolic Learning method. Section 2.2 discusses utilizing LLM to implement interpretable programs, including a connection between prompted-LLM and interpretable unit (Section 2.2.1), the Domain Specific Language (Section 2.2.2) and learning algorithm (Section 2.2.3).

2.1 Preliminaries on Neurosymbolic Learning

NeuroSymbolic Programmings (NSPs) [Chaudhuri et al., 2021, Shah et al., 2020, Cui and Zhu, 2021, Frosst and Hinton, 2017] seek to couple classical symbolic methods with contemporary neural networks to build expressive and interpretable models. The learning method for NSPs is often characterized by (1) a **Domain Specific Language (DSL)** that specifies available operations of the program and (2) **a learning algorithm** for finding the best instance. The resulting programs are structured, neuro-symbolic terms that follow the syntax specified by the DSL.

Domain-Specific Language (DSL) DSL in NSPs comprises manually defined operators, including interpretable symbolic (e.g. `if-then-else`) and expressive neural components (e.g. `cnn(x, θ)`). These operators can be chained to construct various tree-structured programs, a.k.a. computation graphs. Eq (1) presents an example DSL used to construct the program for predicting the creature species in Figure 1. Here, x and c represents inputs and constants, and α denotes a sub-program:

$$\alpha ::= x \mid c \mid \texttt{Add}(\alpha_1, \alpha_2) \mid \texttt{Mul}(\alpha_1, \alpha_2) \mid \text{If } \alpha_1 \text{ Then } \alpha_2 \text{ Else } \alpha_3 \mid \texttt{cnn}(x, \theta) \mid \texttt{Dist}(\alpha_1, \alpha_2). \quad (1)$$

Co-optimization of program structure and learnable parameters In NSPs, the construction of a program involves solving a combinatorial optimization problem for both the program structure and the parameters of its learnable operators (e.g. neural components). As the number of DSL operators increases, the complexity of this task grows exponentially. To make the search process more tractable, existing research employs various approximation techniques to efficiently identify viable candidates, including greedy tree search [Shah et al., 2020], continuous relaxation [Cui and Zhu, 2021], distillation [Frosst and Hinton, 2017] and meta-learning [Chaudhuri et al., 2021].

Limitations While the integration of symbolic and neural components in NSPs represents a promising innovation, the incorporating of neural modules inevitably introduces black-box components and makes the program non-interpretable. Researchers have attempted to address this issue through two primary approaches: restricting the DSL to only interpretable operators [Shah et al., 2020, Cui and Zhu, 2021], or employing prototype learning to derive relatively interpretable neural modules [Nauta et al., 2021b, Ming et al., 2019, Nauta et al., 2021a]. However, the DSL approach is not automatic, heavily relies on domain expertise, and potentially overlooking crucial information not identified by experts; Conversely, prototype learning aims to represent the concept of each neural module by a set of representative samples, which is not guaranteed to success.

2.2 LLM-Symbolic Programs

This section explores how LLMs can effectively be utilized to implement NSPs' modules that are expressive, interpretable, and straightforward to learn with LLMs.

2.2.1 Prompted-LLM as an interpretable unit

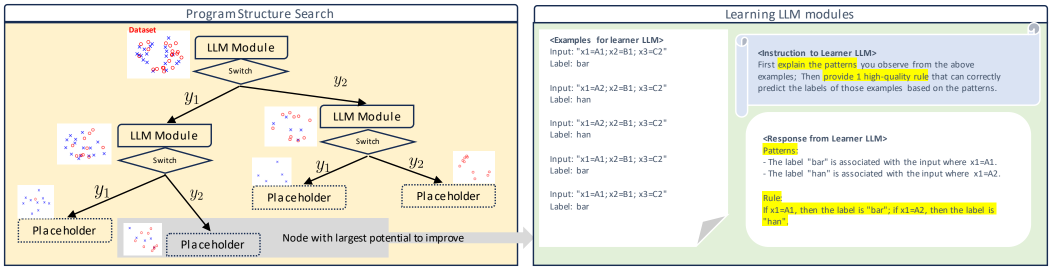
The trade-off between interpretability and expressiveness presents a fundamental limitation in machine learning. Machines perceive images and text as raw binary signals, and transforming these into interpretable concepts inevitably requires complex and non-interpretable components, such as neural networks. Even human perception remains non-interpretable, as we lack a complete understanding of how the brain processes signals. However, The following analysis suggests that pretrained LLM offer a potential avenue to bridge this gap.

Connection between interpretable learning and prompting LLMs pretrained on the next-token prediction task model the following joint distribution of a sequence of tokens $\{w_t\}_{t=1}^{T}$

$$P(w_1, w_2, \dots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_{t-1}, w_{t-2}, \dots, 1) = f_\theta(w_t \mid w_1, w_2, \dots, w_{t-1}),$$

where the conditional probabilities are parameterized by an auto-regressive model $f(\cdot; \theta)$ (e.g. Transformer) and each word w_t is predicted given all the preceding tokens. The pretraining objective minimizes the following negative log-likelihood:

$$\min_\theta \mathcal{L}(\theta) = -\sum_{t=1}^{T} \log f_\theta(w_t \mid w_{t-1}, \dots, w_1). \quad (2)$$



<Examples for learner LLM>
Input: "x1=A1; x2=B1; x3=C2"
Label: bar

Input: "x1=A2; x2=B1; x3=C2"
Label: han

Input: "x1=A2; x2=B1; x3=C2"
Label: han

Input: "x1=A1; x2=B1; x3=C2"
Label: bar

Input: "x1=A1; x2=B1; x3=C2"
Label: bar

<Instruction to Learner LLM>
First explain the patterns you observe from the above examples; Then provide 1 high-quality rule that can correctly predict the labels of those examples based on the patterns.

<Response from Learner LLM>
Patterns:
- The label "bar" is associated with the input where x1=A1.
- The label "han" is associated with the input where x1=A2.

Rule:
If x1=A1, then the label is "bar"; if x1=A2, then the label is "han".

Figure 2: **Learning Algorithm for LSPs.** The learning algorithm for LSPs contains two parts: **(1) program structure search (Left):** This process is akin to constructing a traditional decision tree. Starting from the root, the algorithm traverses down the tree, iteratively splitting the training dataset based on the current node's predictions and expanding the leaf node with the highest prediction errors. **(2) LLM module optimization (Right):** Here, a learner LLM is instructed to summarize rules based on the observed data at its node.

A key observation from Eq. (2) is that the training process optimizes a "SuperNet" of conditional probabilistic models (CPM), each defined by an instruction s: $f_{s,\theta}(y|x) = f_\theta(y \mid x, s)$, where x is the input and s is the instruction for a particular task. Therefore, with a fixed LLM, the set of natural language prompts, denoted as $\mathcal{S}$, provides a massive set of interpretable neural network modules for the task. For a given dataset $\{(x_i, y_i)\}_{i=1}^n$, finding the best prompt to minimize the empirical loss, $\min_{s\in\mathcal{S}} \sum_{i=1}^n \mathcal{L}((f_{s,\theta}(y_i \mid x_i)))$, can be viewed as a form of learning, and the resulting model is inherently interpretable, as the prompt s is expressed in natural language.

This connection reveals that prompt optimization within the natural language space offers a form of interpretable learning that simultaneously achieves both expressiveness and interpretability. The key to bridging this gap lies in leveraging LLMs to handle the non-interpretable processing of raw signals into high-level concepts, much like how neurons in the human brain transform signals into information. This allows learning to occur within an interpretable space.

Limitation of (discrete) prompt optimization However, existing prompt optimization algorithms are insufficient for interpretable learning for several reasons: firstly most methods focus on "rewriting" prompts to enhance performance [Pryzant et al., 2023, Hsieh et al., 2023], which might not help in extracting interpretable knowledge from data. Additionally, while recent developments show some capabilities in correcting prompts using error examples [Pryzant et al., 2023, Wang et al., 2023], they still struggle with complex decision rules, such as conditional branching for classification tasks. These rules, often applicable to only a subset of samples, are difficult to recover when considering the whole training set. Our experiments indicate that direct application of current methods fails to effectively address complex decision rules. These challenges motivate the proposed LSP framework that combines prompt optimization with symbolic programs.

2.2.2 Domain-Specific Language of LSPs

Compared with traditional NSPs that require manually designing a comprehensive DSL, LLM's ability to represent a wide range of functions via different prompting, we can significantly streamline the grammar required to build expressive and interpretable models. Specifically, for predictive models, we can build powerful LSPs from a minimalist DSL with only three components: the input, conditional branching, and LLM module:in

$$\alpha ::= x \mid \texttt{switch}(\{\alpha == y_i : \alpha_i\}_{i=1}^k) \mid \texttt{LLM}(x, s). \tag{3}$$

Here, **input** x represents the input data (text, image, etc); the **conditional branching** $\texttt{switch}(\{y_i : \alpha_i\}_{i=1}^k)$ forms the backbone of the program structure. Each switch can be viewed as a node in a decision tree tree with k branches. It will branch to α_i if the sub-program α predicts y_i. **The LLM Module** $\texttt{LLM}(x, s)$ serves as the inference engines. It means to prompting LLM to make a prediction on input x under the instruction s.

Figure 1 shows an example LSP generated from above DSL. Given a test query, we traverse the tree-structured program in a top-down manner, assigning data to specific child node based on the parent node's predictions, until the leaf node is reached and the final response is returned.

2.2.3 Learning algorithm

Overview The tree search framework commonly used in NSPs also applies to LSPs [Chaudhuri et al., 2021, Shah et al., 2020]. Moreover, the simplicity of our DSL allows the search process to be further arranged in a highly intuitive manner akin to constructing decision trees. As illustrated in

Figure 2, the process begins at the root node with an empty program and the entire training set. A `switch` operator combined with an `LLM(x, s)` module is then added. This combination essentially directs the program's flow based on the module's predictions. Once the LLM module is trained, we expand its child nodes and repeat the whole process. This divide-and-conquer strategy benefits the search process by simplifying each LLM module's task to fitting only a subset of the data.

Learning LLM-modules by summarizing predictive rules In LSPs, each LLM module is responsible for decision-making on its designated data subset. For traditional NSPs, the neural modules are optimized using empirical risk minimization; For LSPs, training LLM modules essentially transforms into deriving rules from observed data, as established in Section 2.2.1. While this can be achieved via generic prompt optimization techniques, we adopt a more direct approach utilizing the LLM's robust summarization capabilities [Adams et al., 2023, Goyal et al., 2022, Zhang et al., 2024, Pu and Demberg, 2023], asking the model to summarize rules from observed data patterns. Note that LLM serves as both the inference engine and the learner, as depicted in Figure 2 (Right).

Node selection Top-down tree search algorithms generally use a node scoring function to determine the next node to expand, ideally prioritizing nodes with the greatest potential for program improvement. Given that nodes with more frequent errors likely have more room for improvement, we use error count as our scoring function. This metric, accounting for both the error rate and the size of the data subset each node handles, provides a simple yet empirically effective approach. Section 6 presents empirical evidence supporting the efficacy and robustness of this metric.

Complete algorithm The above outlines the learning process for expanding a single program. In the full search pipeline, we further incorporate beam search [Pryzant et al., 2023] and batch sampling to prevent the search from getting stuck in local minima, as summarized in Algorithm 2 (Appendix).

3 IL-Bench Interpretable-Learning Benchmark

The goal for interpretable learning is for the model to acquire knowledge transferable to humans for classification tasks that are NOT zero-shot solvable. A task is zero-shot solvable if the LLM can predict accurately based solely on the class name (e.g. basic dog-cat classification). To evaluate interpretable learning methods, we need classification tasks with classes unseen during LLM pretraining, requiring the model to learn additional knowledge for successful classification. This is challenging given the extensive pretraining on Internet data. Therefore, we introduce Interpretable-Learning Bench, a novel benchmark comprising multiple challenging tasks, and even advanced LLMs like GPT need to learn substantial additional knowledge to solve these tasks. This benchmark provides a valuable resource for evaluating future interpretable learning and autoprompting methods. We will explain the construction of IL-Bench next, and leave the detailed summaries to Appendix Table 7.

Synthetic Tasks Prior work in symbolic learning often uses synthetic datasets to evaluate methodologies due to known oracle rules, making it easy to observe model performance. Inspired by this, we develop synthetic datasets with known predictive rules in our benchmark. These datasets employ symbols to represent variables and values, making them context-agnostic and requiring the model to rely on abstract reasoning. Our tasks include decisions generated by decision trees of varying complexity, allowing us to assess model behavior as rule complexity increases.

Textual Classification Tasks: from image to text dataset To evaluate the model's proficiency in complex scenarios, Fine-Grained Visual Classification (FGVC) tasks [Maji et al., 2013, Wah et al., 2011, Kramberger and Potočnik, 2020, Nilsback and Zisserman, 2008, Van Horn et al., 2015] serve as an excellent testbed. FGVC focuses on distinguishing objects within narrowly defined categories, where the differences between examples from each class are often subtle. We selected specific image subsets from FGVC databases, covering area of various bird species [Wah et al., 2011]. For evaluating pure-text LLMs, we convert them to text datasets via captioning. To ensure the captions cover the intricate differences between objects, we introduce **the Concatenated Contrastive Captioning (C3) technique**, which generates detailed captions by contrasting a target image with reference images from different classes, merging them into a comprehensive description for text-only LLM evaluation.

Visual classification Tasks: distinguishing novel visual concepts We also collect a new suit of datasets from Palworld, a Pokemon-style game containing various species of creatures (Examples in

Table 7). Since the game is released after most existing LLM's knowledge cut-off date, the model will need to rely solely on the knowledge extracted from the dataset to perform the prediction.

4 Related Work

Interpretable machine learning Although neural networks are immensely expressive, they provide no insights into its internal decision making mechanism. In the quest of making model predictions interpretable, research has broadly categorized methods into two main types: post-hoc and intrinsic. Post-hoc methods provide insights into how a pretrained model behaves, usually by highlighting important features used for decision making [Zintgraf et al., 2017, Petsiuk et al., 2018, Dabkowski and Gal, 2017, Shrikumar et al., 2017, Sundararajan et al., 2017, Ancona et al., 2017] or provide counterfactual explanations [Dhurandhar et al., 2018, Hendricks et al., 2018, van der Waa et al., 2018, Goyal et al., 2019, Hsieh et al., 2021]. Beyond attribution in the feature space, some methods can also be generalized to the space of higher level concepts [Kim et al., 2018, Bai et al., 2023]. However, all these methods aim to highlight important features while not being able to recover the entire decision making process of neural networks.

On the other hand, intrinsic methods integrate interpretability directly into the model's architecture, making them naturally interpretable by design. Traditional Methods include Decision Trees [Chen and Guestrin, 2016] and Generalized Additive Models (GAMs) [Hastie and Tibshirani, 1990] offer strong interpretability, yet often not expressive enough. Concept bottleneck model adds a hidden layer in neural network, where neurons represent some predefined concepts to gain interpretability [Koh et al., 2020, Losch et al., 2019, Yuksekgonul et al., 2022, Oikarinen et al., 2023]. While this approach facilitates attribution of concepts, it does not provide a comprehensive decision rule. Neurosymbolic Programming (NSP) [Chaudhuri et al., 2021, Shah et al., 2020, Cui and Zhu, 2021, Nauta et al., 2021b] represents an innovative blend, combining deep learning's data handling capabilities with symbolic reasoning to foster both performance and transparency. Despite early promises, NSP suffers from an inherit trade-off between expressiveness (more NN modules) and interpretability (more symbolic modules). Moreover, they are often expensive to train due to co-optimization of program architecture and parameters of the NN modules [Shah et al., 2020, Cui and Zhu, 2021].

Prompt Optimization The essence of utilizing a generative language model lies in crafting effective prompts. Recent advancements have aimed to automate this process, reducing the need for human effort through prompt optimization [Shin et al., 2020, Zhou et al., 2022]. While pioneering efforts were mainly directed towards various discrete optimization algorithms [Shin et al., 2020, Deng et al., 2022, Zhang et al., 2022], it has been noted that advanced LLMs can revise prompts similarly to human engineers [Zhou et al., 2022, Pryzant et al., 2023]. Since these initial efforts, a significant body of research has emerged, exploring various search algorithms including Monte Carlo Sampling [Zhou et al., 2022], beam search [Pryzant et al., 2023], evolutionary search [Yang et al., 2023, Fernando et al., 2023, Xu et al., 2022, Guo et al., 2023, Hsieh et al., 2023], and tree search [Wang et al., 2023]. However, existing methods often treat the prompt as a single entity without explicit structure. From this perspective, prompt optimization methods can be seen as simplified instances of LSPs, where the program consists solely of one LLM module. While this simplification has shown promising results, as task complexity increases, the explicit structuring within LSPs allows them to encode knowledge from data. This provides substantial advantages over conventional prompt optimization methods.

5 Experimental Results

We adopt a comprehensive approach to extensively evaluate the effectiveness of LSPs against various baselines under different settings. Our empirical study is designed to validate the benefits of LSPs over alternative methods by addressing the following research questions:

- **Q1: How does LSP compare against traditional NSPs in expressiveness and interpretability?** We assess this through both quantitative and qualitative evaluations (human studies). (Section 5.2)
- **Q2: Does LSP generalize better than traditional NSPs under domain shifts?** This question is explored in detail in Section 5.2.
- **Q3: Is the incorporation of explicit structures beneficial to LSPs?** We compare the structured LSP with vanilla prompt optimization, which exemplifies a special case of LSP with a single LLM module. (Section 5.3)

Table 1: **Classification accuracy comparison with XAI methods on IL-Bench-Vision.** Here, all numbers for LSP are obtained with Gemini-Vision as the learner and inference LLM, except for LSP (GPT-4V) which uses the larger GPT-4V as the learner; Decision Tree, operating directly on pixel data, lacks human interpretability. Key findings include: (1) Our method outperforms XAI baselines with an average accuracy of 95.67%, which is over 10% higher than the nearest competitor. (2) The program generated by LSP also demonstrates superior transferability to human raters, as they are able to reproduce the predictions following rules learned by LSP.

IL-Bench-Vision		Palworld							
MLLM	Method	Mean	Fire-1	Fire-2	Dragon-1	Dragon-2	Electric-1	Electric-2	Water-1
Gemini-M	Decision Tree [Chen and Guestrin, 2016]	68.20	91.11 ± 12.57	32.00 ± 9.80	68.33 ± 10.27	48.33 ± 20.95	82.67 ± 6.80	65.33 ± 13.60	66.67 ± 8.50
	ProtoTree [Nauta et al., 2021b]	84.33	**100.00 ± 0.00**	62.67 ± 12.36	**98.33 ± 2.36**	85.00 ± 4.08	**100.00 ± 0.00**	82.67 ± 9.98	61.67 ± 25.93
	LSP	84.76	93.33 ± 6.67	76.00 ± 4.00	85.00 ± 0.00	**97.50 ± 2.50**	86.00 ± 10.00	68.00 ± 4.00	87.50 ± 2.50
	LSP (GPT-4V)	**95.67**	96.67 ± 3.33	**90.00 ± 6.00**	90.00 ± 10.00	**97.50 ± 2.50**	**100.00 ± 0.00**	**98.00 ± 2.00**	**97.50 ± 2.50**
Human Rater	ProtoTree [Nauta et al., 2021b]	72.74	83.33 ± 16.67	50.0 ± 10.0	**100.0 ± 0.0**	75.0 ± 0.0	83.33 ± 16.67	80.0 ± 0.0	37.5 ± 12.5
	LSP (GPT-4V)	**90.36**	**100.00 ± 0.00**	**70.00 ± 10.00**	**100.00 ± 0.00**	**87.5 ± 12.5**	**100.00 ± 0.00**	**100.00 ± 0.00**	**75.00 ± 25.00**

- **Q4: How effective are different LLMs in implementing LSP?** We conduct cross-model experiments to evaluate the performance of various LLMs as the computational backbone for learning and inference in LSP. (Section 8.1.1)

5.1 General settings

Evaluation For language tasks, we test popular LLMs, including GPT-3.5 (`turbo-1104`) [Ouyang et al., 2022], GPT-4 (`1106-preview`) [Achiam et al., 2023], and Gemini-M (`1.0-pro`) [Team et al., 2023]. For vision tasks, GPT-4V (`gpt-4-1106-vision-preview`) and Gemini-Vision (`1.0-pro-vision`) are utilized. All experiments are repeated with 3 seeds.

Implementation details of LSP Our default model of choice is GPT-3.5 for language tasks and Gemini-Vision for vision tasks, but also examine cross-(M)LLM performance in Appendix. All LLM modules are initialized with an empty instruction "none". More detailed hyperparameters can be found in Appendix, which is kept fixed throughout the experiments.

5.2 Comparison with traditional interpretable learning methods

We compare LSP with two established models - ProtoTree [Nauta et al., 2021b] and Decision Tree [Chen and Guestrin, 2016] - both organize prediction process in tree-structured formats. Among existing NSP methods, the closest to ours is ProtoTree - a highly interpretable NSP that learns a discrete binary tree end-to-end, where each node stores an image patch ("prototype") and the edges determine whether the prototype exists within the query image. Note that ProtoTree does not rely on an explicit DSL - we could not compare with methods based on explicit DSL since they require domain experts to design those operation, while our goal is to automate the whole process. Since ProtoTree only implements image tasks, this comparison also focus on the vision tasks in IL-Bench.

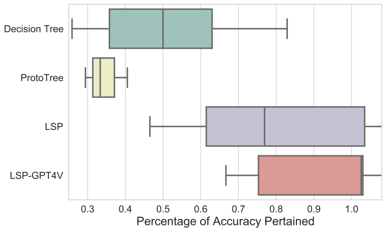


Figure 3: **Accuracy retention rate on Out-Of-Distribution variants of IL-Bench-Vision testsets.** We compute the ratio of test accuracy evaluated on OOD datasets to the original test accuracy. LSP shows strong transferability to OOD data. Notably, the version using GPT-4V as the learner retains 90-100% of the original test accuracy.

Expressiveness The expressiveness of the learned programs is evaluated in Table 1. LSP (GPT4V) outperforms ProtoTree with an average accuracy of 95.67%, which is over 10% gain. Considering that GPT/Gemini has never observed the images in our datasets before (curated after their knowledge cutoff), this result suggests LSP is capable of formulating effective predictive rules from previously unseen examples.

Interpretability We measure the interpretability of LSPs and NSPs by having human raters make predictions based on visualizations of the learned programs (See Appendix for evaluation protocols). This process essentially "transfers" knowledge from models back to human. Notably, many XAI methods fall short of achieving this level of interpretability, with ProtoTree being a rare exception. As summarized in Table 1, the program generated by LSP also demonstrates stronger transferability to human raters, as they are able to largely reproduce the predictions following rules learned by LSP.

Table 2: **Classification accuracy comparison with Prompt Optimization methods on IL-Bench-Language.** Key findings include: (1) LSP achieves $\sim 5\%$ accuracy gain over PromptAgent, the previous state-of-the-art. (2) Across synthetic Decision Tree datasets categorized by increasing complexity of oracle decision rules (Easy, Medium, Hard), LSP consistently outperforms other methods in maintaining high accuracy levels, demonstrating its superior ability to reverse-engineer complex rules from observed data.

Text Benchmark			Synthetic			Caption					
LLM	Method	Mean	DT-Easy	DT-Medium	DT-Hard	Waxwing	Waterthrush	Jaeger	Albatross	Blackbird	Swallow
GPT-3.5	APE [Zhou et al., 2022]	67.62	99.67 ± 0.47	86.67 ± 12.47	87.00 ± 8.57	46.11 ± 4.37	43.89 ± 3.14	65.56 ± 2.83	47.41 ± 2.28	78.06 ± 1.04	54.17 ± 1.18
	OPRO [Yang et al., 2023]	55.48	50.00 ± 1.08	50.17 ± 3.06	30.33 ± 2.62	57.22 ± 2.08	57.22 ± 4.16	76.67 ± 4.71	40.37 ± 3.43	78.06 ± 2.83	55.28 ± 1.04
	APO [Pryzant et al., 2023]	70.67	**100.00** ± **0.00**	96.67 ± 4.71	77.83 ± 11.90	56.11 ± 4.78	48.89 ± 4.16	70.00 ± 5.93	54.07 ± 9.70	74.17 ± 2.97	58.33 ± 1.36
	PromptAgent [Wang et al., 2023]	72.40	97.67 ± 3.30	88.50 ± 8.44	64.33 ± 20.27	60.56 ± 4.78	56.67 ± 6.24	75.00 ± 3.60	**74.44** ± **6.54**	74.17 ± 1.36	57.22 ± 0.79
	LSP (Ours)	**77.29**	99.25 ± 0.75	**98.50** ± **0.00**	**89.75** ± **1.25**	**65.83** ± **4.17**	**62.50** ± **0.83**	**80.00** ± **1.67**	61.11 ± 1.11	**78.75** ± **0.42**	**62.92** ± **0.42**

Table 3: **Classification accuracy comparison with Prompt Optimization methods on IL-Bench-Vision.** LSP achieves an average accuracy of 84.47%, which is $\sim 20\%$ higher than the 2nd best method (APE).

Vision Benchmark		Palworld							
MLLM	Method	Mean	Fire-1	Fire-2	Dragon-1	Dragon-2	Electric-1	Electric-2	Water-1
Gemini-M	APE [Zhou et al., 2022]	64.38	76.67 ± 3.33	56.00 ± 0.00	59.17 ± 5.83	35.00 ± 5.00	57.33 ± 10.67	**82.00** ± **2.00**	82.50 ± 12.50
	OPRO [Yang et al., 2023]	49.26	86.67 ± 0.00	32.00 ± 4.00	56.67 ± 3.33	50.00 ± 5.00	48.00 ± 28.00	49.00 ± 19.00	27.50 ± 2.50
	APO [Pryzant et al., 2023]	58.45	76.67 ± 23.33	42.00 ± 14.00	59.17 ± 5.83	70.00 ± 5.00	39.33 ± 7.33	67.50 ± 12.50	52.50 ± 17.50
	PromptAgent [Wang et al., 2023]	54.74	66.67 ± 6.67	44.00 ± 0.00	49.17 ± 15.83	52.50 ± 2.50	55.33 ± 8.67	42.50 ± 2.50	70.00 ± 0.00
	LSP (Ours)	**84.47**	**93.33** ± **6.67**	**76.00** ± **4.00**	**85.00** ± **0.00**	**97.50** ± **2.50**	**86.00** ± **10.00**	68.00 ± 4.00	**87.50** ± **2.50**

Generalization under Domain Shift In contrast to traditional NSP models that rely on parametric memory, LSP utilizes language instructions to encode knowledge. This strategy significantly enhances robustness against variations in visual attributes (domain shifts). To verify this advantage, we examine the transferability of the learned programs to Out-of-Distribution (OOD) data, constructed using GPT-4V (See Appendix for details) As shown in Figure 3, LSP demonstrates exceptional resilience to domain shifts, compared with ProtoTree.

5.3 Comparison with prompt optimization

Since there exists a variety of PO method that primarily differ in the search algorithm, we select one most representative method from each major category: Monte Carlo sampling (APE) [Zhou et al., 2022], evolutionary search (ORPO) [Yang et al., 2023], beam search (APO) [Pryzant et al., 2023], and tree search (PromptAgent) [Wang et al., 2023]. Since the main bottleneck for PO methods is the candidate evaluation, we follow existing works and set the same maximum number of candidate proposals for all methods (100 candidates).

Results The empirical results indicate that incorporating explicit structures significantly enhances performance of the programs on predictive tasks: LSP consistently outperforms all vanilla prompt optimization methods, with a considerable margin of 20.09% and 4.89% over the 2nd best methods on vision and language tasks respectively. The advantages of integrating structured learning are twofold: (1) It simplifies the learning process: LSP benefits from a divide-and-conquer approach where each LLM-module node focuses solely on extracting predictive rules for a specific subset of the data. (2) It streamlines the inference process: We observe that LLMs tend to exhibit hallucination as the complexity of the instructions increases (e.g., multiple conditional clauses. In contrast, LSP mitigates this issue by ensuring that each LLM module contains simpler, more manageable instructions.

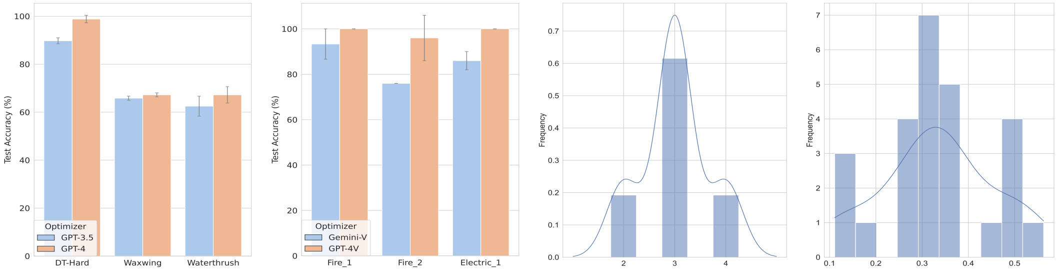
6 Ablation Study

Convergence of LLM-Symbolic Program LSP LSP organizes instructions into a tree-based structure. Such divide-and-conquer strategy simplifies the learning process. To verify this, we also plot the training trajectories for LSP across various tasks. The training trajectory indicates the how fast a model fits the observed examples. As Figure 5 demonstrates, LSP not only converges faster but also achieves higher final accuracy compared to models that use unstructured prompting techniques.

Search cost analysis We also report the actual runtime of search and inference process for different methods in Table 4.

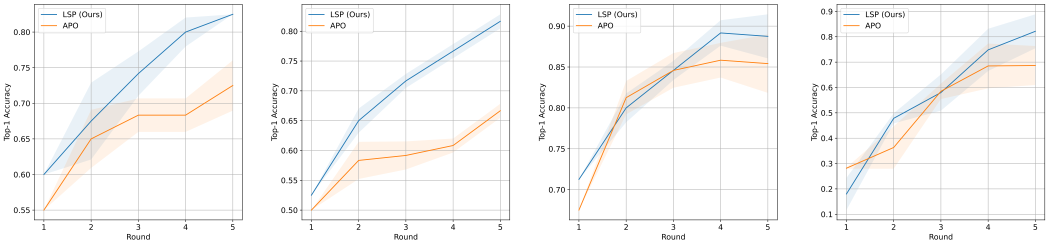
Table 4: **Search and inference runtime Ccmparison.** Despite the multi-step decision-making process of LSP's tree structure, our method incurs comparable search and inference costs to various prompt optimization baselines.

Method	Search (s)	Inference (s)
APE	270.60	0.11
OPRO	257.86	0.14
APO	270.85	**0.08**
PromptAgent	**220.95**	0.11
LSP	232.54	0.13



(a) Language Tasks (b) Vision Tasks (c) Program Depth (d) Program Sparsity

Figure 4: **(a, b): Stronger LLMs as better LSP learners.** In these experiments, we keep the inference LLM fixed (GPT-3.5 for text and Gemini-V for images) while swapping the learner LLM with GPT-4. With its larger parameter count, GPT-4 consistently achieves better performance in learning LSPs. **(c, d): Statistics of discovered programs.** Averaged from the IL-Bench-Language tasks, the resulting LSPs are generally shallow and sparse, indicating that the final prediction can be reached within only a few steps.



(a) CUB-Waxwing (b) CUB-Waterthrush (c) CUB-Blackbird (d) DT-Hard

Figure 5: **Convergence of different algorithms across time**. We plot the trajectory of training accuracy against the number of optimization rounds. The API model is GPT-3.5. (1). LSP converges substantially faster than vanilla prompting; (2). The search process does not introduce extra variances.

We found that LSP incurs comparable search and inference costs to various prompt optimization baselines.

Different node scoring functions Table 5 summarizes the performance of LSP using three different node scoring functions: (1). Error count. (2). Prediction accuracy. (3). Random scoring. The results suggest that Error count achieves substantially better and more consistent outcomes across different tasks.

Table 5: **Comparison of Different Node Scoring Functions** on three tasks from IL-Bench-Language. Despite its simplicity, error count achieves more consistent performance compared to alternative metrics.

Node Scoring	DT-Hard	Waxwing	Waterthrush
Random	70.50 ± 11.01	62.22 ± 4.78	61.67 ± 1.36
Accuracy	80.33 ± 18.27	**66.11 ± 7.86**	54.44 ± 0.70
Error Count (LSP)	**89.75 ± 1.25**	65.83 ± 4.17	**62.50 ± 0.83**

Complexity of Learned LSP Our analysis of the statistics of learned programs indicates that the complexity of programs developed by LSP is quite manageable: Most programs can reach a final prediction within just three steps, as illustrated in Figure 4c, and the tree structures tend to be sparse, as shown in Figure 4d. These observations confirm that although theoretical maximum tree expansion could grow exponentially with depth, in practice, LSPs operate effectively without requiring overly complex structures. This further explain why LSP exhibits the reasonable search and inference costs documented in Table 4.

7 Conclusion

This work aims at revitalizing the concept of Neuro-Symbolic Programming in the era of Large Language Models. We demonstrate that pretrained LLMs can implement powerful symbolic programs that are expressive, interpretable, and easy to train. Additionally, we introduce the Instruction Learning Benchmark (IL-Benchmark), which consists of a suite of vision and language datasets designed to evaluate instruction learning algorithms. We hope that our proposed framework will inspire new developments in interpretable learning methods during the LLM era. We regard our study as an initial step in the research on LLM-Symbolic Programs. Accordingly, we acknowledge the **limitations** of the current method in Appendix Section 8.4.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. arXiv preprint arXiv:2303.08774, 2023.

Griffin Adams, Alexander Fabbri, Faisal Ladhak, Eric Lehman, and Noémie Elhadad. From sparse to dense: Gpt-4 summarization with chain of density prompting. arXiv preprint arXiv:2309.04269, 2023.

Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. Towards better understanding of gradient-based attribution methods for deep neural networks. arXiv preprint arXiv:1711.06104, 2017.

Andrew Bai, Chih-Kuan Yeh, Pradeep Ravikumar, Neil YC Lin, and Cho-Jui Hsieh. Concept gradient: Concept-based interpretation without linear assumption. In ICLR, 2023.

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.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, pages 785–794, 2016.

Guofeng Cui and He Zhu. Differentiable synthesis of program architectures. Advances in Neural Information Processing Systems, 34:11123–11135, 2021.

Piotr Dabkowski and Yarin Gal. Real time image saliency for black box classifiers. In Advances in Neural Information Processing Systems, pages 6967–6976. NeurIPS, 2017.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P Xing, and Zhiting Hu. Rlprompt: Optimizing discrete text prompts with reinforcement learning. arXiv preprint arXiv:2205.12548, 2022.

Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Paishun Ting, Karthikeyan Shanmugam, and Payel Das. Explanations based on the missing: Towards contrastive explanations with pertinent negatives. In Advances in Neural Information Processing Systems, pages 592–603. NeurIPS, 2018.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. arXiv preprint arXiv:2309.16797, 2023.

Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. arXiv preprint arXiv:1711.09784, 2017.

Tanya Goyal, Junyi Jessy Li, and Greg Durrett. News summarization and evaluation in the era of gpt-3. arXiv preprint arXiv:2209.12356, 2022.

Yash Goyal, Ziyan Wu, Jan Ernst, Dhruv Batra, Devi Parikh, and Stefan Lee. Counterfactual visual explanations. In International Conference on Machine Learning, pages 2376–2384. ICML, 2019.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. arXiv preprint arXiv:2309.08532, 2023.

Trevor Hastie and Robert Tibshirani. Generalized additive models. Chapman and Hall/CRC, 1990.

Lisa Anne Hendricks, Ronghang Hu, Trevor Darrell, and Zeynep Akata. Grounding visual explanations. In ECCV. ECCV, 2018.

Cheng-Yu Hsieh, Chih-Kuan Yeh, Xuanqing Liu, Pradeep Kumar Ravikumar, Seungyeon Kim, Sanjiv Kumar, and Cho-Jui Hsieh. Evaluations and methods for explanation through robustness analysis. In International Conference on Learning Representations. ICLR, 2021. URL https://openreview.net/forum?id=4dXmpCDGNp7.

Cho-Jui Hsieh, Si Si, Felix X Yu, and Inderjit S Dhillon. Automatic engineering of long prompts. arXiv preprint arXiv:2311.10117, 2023.

Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, et al. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In International Conference on Machine Learning, pages 2673–2682. ICML, 2018.

Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In International conference on machine learning, pages 5338–5348. PMLR, 2020.

Tin Kramberger and Božidar Potočnik. Lsun-stanford car dataset: enhancing large-scale car image datasets using deep learning for usage in gan training. Applied Sciences, 10(14):4913, 2020.

Max Losch, Mario Fritz, and Bernt Schiele. Interpretability beyond classification output: Semantic bottleneck networks. arXiv preprint arXiv:1907.10882, 2019.

Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. arXiv preprint arXiv:1306.5151, 2013.

Yao Ming, Panpan Xu, Huamin Qu, and Liu Ren. Interpretable and steerable sequence learning via prototypes. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 903–913, 2019.

Meike Nauta, Annemarie Jutte, Jesper Provoost, and Christin Seifert. This looks like that, because... explaining prototypes for interpretable image recognition. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pages 441–456. Springer, 2021a.

Meike Nauta, Ron Van Bree, and Christin Seifert. Neural prototype trees for interpretable fine-grained image recognition. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 14933–14943, 2021b.

Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In 2008 Sixth Indian conference on computer vision, graphics & image processing, pages 722–729. IEEE, 2008.

Tuomas Oikarinen, Subhro Das, Lam M Nguyen, and Tsui-Wei Weng. Label-free concept bottleneck models. arXiv preprint arXiv:2304.06129, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. Advances in neural information processing systems, 35:27730–27744, 2022.

Pocket Pair. Palworld, 2024. URL https://en.wikipedia.org/wiki/Palworld.

Vitali Petsiuk, Abir Das, and Kate Saenko. Rise: Randomized input sampling for explanation of black-box models. arXiv preprint arXiv:1806.07421, 2018.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with" gradient descent" and beam search. arXiv preprint arXiv:2305.03495, 2023.

Dongqi Pu and Vera Demberg. Chatgpt vs human-authored text: Insights into controllable text summarization and sentence style transfer. arXiv preprint arXiv:2306.07799, 2023.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pages 1135–1144. ACM, 2016.

Ameesh Shah, Eric Zhan, Jennifer Sun, Abhinav Verma, Yisong Yue, and Swarat Chaudhuri. Learning differentiable programs with admissible neural heuristics. Advances in neural information processing systems, 33:4940–4952, 2020.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. arXiv preprint arXiv:2010.15980, 2020.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. International Conference on Machine Learning, 2017.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In International Conference on Machine Learning, pages 3319–3328. PMLR, 2017.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. arXiv preprint arXiv:2312.11805, 2023.

Jasper van der Waa, Marcel Robeer, Jurriaan van Diggelen, Matthieu Brinkhuis, and Mark Neerincx. Contrastive Explanations with Local Foil Trees. In 2018 Workshop on Human Interpretability in Machine Learning (WHI). WHI, 2018.

Grant Van Horn, Steve Branson, Ryan Farrell, Scott Haber, Jessie Barry, Panos Ipeirotis, Pietro Perona, and Serge Belongie. Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 595–604, 2015.

C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P Xing, and Zhiting Hu. Promptagent: Strategic planning with language models enables expert-level prompt optimization. arXiv preprint arXiv:2310.16427, 2023.

Hanwei Xu, Yujun Chen, Yulun Du, Nan Shao, Yanggang Wang, Haiyu Li, and Zhilin Yang. Gps: Genetic prompt search for efficient few-shot learning. arXiv preprint arXiv:2210.17041, 2022.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. arXiv preprint arXiv:2309.03409, 2023.

Mert Yuksekgonul, Maggie Wang, and James Zou. Post-hoc concept bottleneck models. arXiv preprint arXiv:2205.15480, 2022.

Tianjun Zhang, Xuezhi Wang, Denny Zhou, Dale Schuurmans, and Joseph E Gonzalez. Tempera: Test-time prompting via reinforcement learning. arXiv preprint arXiv:2211.11890, 2022.

Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B Hashimoto. Benchmarking large language models for news summarization. Transactions of the Association for Computational Linguistics, 12:39–57, 2024.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. arXiv preprint arXiv:2211.01910, 2022.

Luisa M Zintgraf, Taco S Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. arXiv preprint arXiv:1702.04595, 2017.

8 Supplemental Material

8.1 Additional ablation experiments

8.1.1 Using different LLMs to implement LSPs

The role of LLMs in LSPs is twofold: they serve both as the inference and learning engine of the LLM-modules in the grammar. The learning engine is responsible for summarizing and organizing patterns from observed data samples into clear predictive rules, whereas the inference engine follows the learned program to make predictions on test examples. Natural questions arise: (1). how effective are different LLMs at optimizing LSPs? (2). Is the learned programs interpretable to different LLMs?

LLM as LSP learner We replace the learning engine used in optimizing LSP with various LLMs - GPT-3.5, Gemini, and GPT-4 - while keeping all other settings consistent with the main experiment. As shown in Figure 4, GPT-4 consistently outperforms other LLMs on both text and vision tasks, while Gemini and GPT-3.5 show similar performance with each other. This reflects their respective capabilities. For specific examples of instructions generated by different LLM optimizers, please see the Appendix.

Table 6: **Transferring LSPs learned from one LLM to another.** The learned LSPs are generally interpretable across various LLMs. However, larger LLMs (e.g., GPT-4) demonstrate a slightly higher consistency in understanding LSPs learned by other LLMs.

Source Model	Task	Evaluator		
		GPT3.5	Gemini-M	GPT4
GPT3.5	DT-Hard	89.75 ± 1.25	72.67 ± 6.91	87.50 ± 1.22
	Waxwing	65.83 ± 4.17	52.22 ± 1.57	56.67 ± 3.60
	Waterthrush	62.50 ± 0.83	64.44 ± 0.79	59.44 ± 3.93
Gemini-M	DT-Hard	75.50 ± 2.04	80.83 ± 1.03	79.17 ± 11.45
	Waxwing	52.78 ± 3.42	58.33 ± 4.91	61.11 ± 10.57
	Waterthrush	50.56 ± 4.16	54.44 ± 5.50	52.22 ± 0.79
GPT4	DT-Hard	74.50 ± 9.35	57.67 ± 3.01	99.50 ± 0.00
	Waxwing	59.44 ± 5.15	62.22 ± 7.49	63.33 ± 4.91
	Waterthrush	66.67 ± 6.80	68.33 ± 2.72	62.78 ± 9.06

LLM as LSP interpreter We then test if LSPs created by one LLM could be interpreted by other LLMs. Table 6 summarizes the performance. The results suggest that LSPs are interpretable across a diverse range of inference models; Larger and stronger LLMs (e.g. GPT-4) demonstrates a slight more consistent ability in interpreting LSPs, which aligns their superior instruction-following capacities.

8.2 Learning algorithm for LSP

The complete pipeline for constructing LSP is summarized in Algorithm 1 and Algorithm 2.

Remarks

- Although initially, the complexity of the program expansion might seem exponential to the tree depth, a closer examination reveals otherwise: (1). In practice, the trees are typically sparse, meaning that expanding only a few branches is often sufficient to achieve good performance (Figure 4d). (2). The divide-and-conquer approach ensures that each tree level processes the same amount of data making the evaluation complexity linear to tree depth.
- The above arrangement of the search process does not compromise generality of LSP: For more sophisticated DSL designs, program structure search can be conducted similarly to traditional NSPs, using top-down tree traversal (cite).

8.3 More details on empirical evaluation

Here we provide more details on the empirical evaluation.

Algorithm 1 `learn_llm_module`: Learning LLM Module by summarizing predictive rules

```
**Input:** Proposal size $m$, data sample $\mathcal{B}$, learner LLM $\mathcal{M}_l$
Initialize an empty list of LLM modules $\Phi$
**for** $i = 1$ **to** $m$ **do**
    Randomly sample $b \sim \mathcal{B}$
    $\phi_{new} \leftarrow$ summarize$(M_l, b)$
    $\Phi \leftarrow \Phi \cup \{\phi_{new}\}$
**end for**
**return** $\Phi$
```

Algorithm 2 Complete pipeline of optimizing LSPs

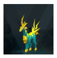
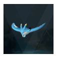
```
**Input:** Dataset $\mathcal{D}$, beam size $d$, number of iterations $T$, inference LLM $\mathcal{M}_i$, learner LLM $\mathcal{M}_l$,
expand ratio $K$, proposal size $m$
Initialize $p_0$ as an empty program
Initialize candidate program set $P = \{p_0\}$
**for** $t = 1$ **to** $T$ **do**
    **for** each program $p$ in $P$ **do**
        ▷ *Batch evaluation*
        Sample a batch $\mathcal{B} \sim \mathcal{D}$
        Evaluate $p$ on $\mathcal{B}$ using $\mathcal{M}_i$
        ▷ *Selecting the most promising node $n$ to expand*
        Assign $\mathcal{B}$ to the leaf nodes of $p$
        Identify the most error-prone leaf node $n$ with assigned subset $\mathcal{B}_n$
        ▷ *Extend program $p$ to $K$ new programs by adding top-$K$ LLM modules to node $n$*
        $\Phi \leftarrow$ learn_llm_module$(n, \mathcal{B}_n, \mathcal{M}_l, m)$
        $\Phi_{topK} \leftarrow$ evaluate and retain top-$K$ $\Phi$ on $\mathcal{B}_n$
        $\mathcal{P}_{new} \leftarrow$ extend $p$ by assigning each $\phi \in \Phi_{topK}$ to node $n$ on program $p$.
        $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}_{new}$
    **end for**
    Evaluate and retain the top-$d$ programs from $\mathcal{P}$ on $\mathcal{D}$
**end for**
**return** The best program from $P$
```

8.3.1 Implementation details

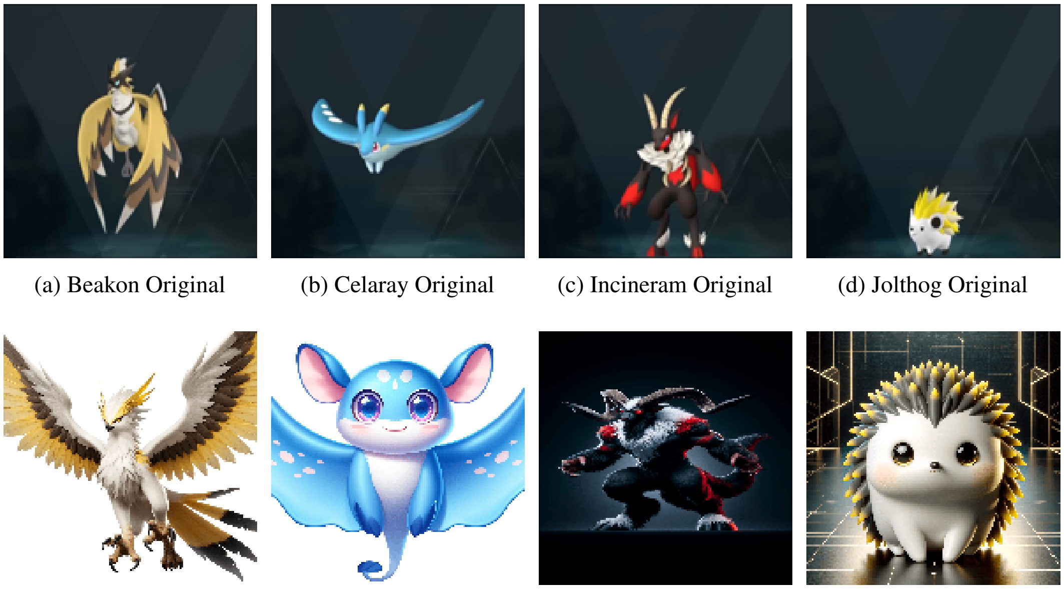
Throughout our main experiment, we use an expansion ratio of 4, batch size of 64, a maximum number of four iterations, and a maximum of 8 candidate (LLM module) proposals for each iteration. The settings for beam search follows that of APO, which uses a beam size of 4 and deploys UCBBandits algorithm with a sample size of 32 to speedup the candidate ranking Pryzant et al. [2023]. The only exception is that for vision tasks, we use a batch size of 4 for cost reduction. The temperature for all API models are set to their default (0.7).

For all prompt optimization baselines, we set the maximum budget (measured by the number of candidate proposals) to the same number. For Decision Tree, we use XGBoost library's standard implementation, which operates on raw pixels. For ProtoTree, we directly run the original implementation, but reduce the maximum depth from 9 to 5, as it is faster to train yet achieves better performance on our datasets. We align the evaluation our baselines

Table 7: **Overview of Interpretable-Learning Benchmark**. We provide task names, types, summaries, number of labels, and one example data point for each task.

Task	Type	Summary	Labels	Example
DT-Easy	Synthetic	Predict labels based on symbolic inputs. Rules generated by a small decision tree	2	"input": "x1=A2; x2=B1", "output": "bar"
DT-Medium	Synthetic	Predict labels based on symbolic inputs. Rules generated by a medium decision tree	2	"input": "x1=A3; x2=B2", "output": "bar"
DT-Hard	Synthetic	Predict labels based on symbolic inputs. Rules generated by a large decision tree	4	"input": "x1=A1; x2=B1; x3=C1", "output": "foo"
Waxwing	Caption	Classify Waxwing species based on its text description.	2	"input": "Tan to light brown head and upper body, black m̈askäcross eyes, lighter cream underparts, bright red tips on secondary wing feathers, small black bill, yellow band on tail.", "output": "Cedar Waxwing"
Waterthrush	Caption	Classify Waterthrush species based on its text description.	2	"input": "Light gray crown, white supercilium, dark eyestripe extending behind eye, olive-brown wings with faint wingbars, white throat, pale underparts, long, slender bill, relatively short tail, orange legs.", "output": "Louisiana Waterthrush"
Jaeger	Caption	Classify Jaeger species based on its text description.	2	"input": "Light greyish-brown plumage on the underside, distinct narrow white band across the nape, wings with a M-shaped pattern when spread, tail slightly forked but mostly straight across.", "output": "Long tailed Jaeger"
Albatross	Caption	Classify Albatross species based on its text description.	3	"input": "Dark brown upperparts and paler brown underparts, elongated and narrow wings with a white trailing edge and distinct finger-like tips, hooked beak with a pale base, light-colored head with a dark eye patch and bill, wings held straight in gliding flight, gliding above water surface. Uniform dark brown plumage, long slender wings, distinct white pattern on underwings, white band near the tips of the underwings, pale or white head, dark eye patch.", "output": "Black footed Albatross"
Blackbird	Caption	Classify Blackbird species based on its text description.	4	"input": "Bright yellow head, black body, sharp conical beak, perched on reed-like vegetation. Bright yellow head, yellow chest, solid black body excluding head and chest, perched on a thin branch. Black body, bright yellow head, sturdy bill, perched on a reed.", "output": "Yellow headed Blackbird"
Swallow	Caption	Classify Swallow species based on its text description.	4	"input": "Light brown head, pale throat, light brown upperparts, long pointed wings, short tail, white underparts, sitting on wire. Light brown head and upper body, white underparts, sitting on a wire, sky background, short beak, sleek body shape. Brown and white plumage, perched on a wire, stout body, short and thick neck, medium-length tail with a straight edge, compact size, unmarked lighter underparts, darker wings and upperparts.", "output": "Bank Swallow"
Fire-1	Vision	Distinguish visually-similar fire-type pals from Palworld.	3	"input": "output:" "Arsox"
Fire-2	Vision	Distinguish visually-similar fire-type pals from Palworld.	5	"input": "output:" "Pyrin"
Dragon-Blue-1	Vision	Distinguish visually-similar blue-colored dragon-type pals from Palworld.	3	"input": "output:" "Elphidran Aqua"
Dragon-Blue-2	Vision	Distinguish visually-similar blue-colored dragon-type pals from Palworld.	4	"input": "output:" "Jetragon"
Electric-1	Vision	Distinguish visually-similar electric-type pals from Palworld.	3	"input": "output:" "Grizzbolt"
Electric-2	Vision	Distinguish visually-similar electric-type pals from Palworld.	4	"input":  "output:" "Univolt"
Water-1	Vision	Distinguish visually-similar water-type pals from Palworld.	4	"input":  "output:" "Celaray"

8.3.2 Constructing Out-Of-Distribution dataset for IL-Bench-Vision tasks



(a) Beakon Original (b) Celaray Original (c) Incineram Original (d) Jolthog Original

(e) Beakon Generated (f) Celaray Generated (g) Incineram Generated (h) Jolthog Generated

Figure 6: **Comparison between original images (top row) and Out-Of-Distribution images (botton row) generated by GPT-4V.** All images are resized to an unified resolution of 128.

Our OOD dataset is constructed by feeding the original image from the training set to GPT-4 (web version), and ask GPT to generate a variant of the input image. The prompt we used is shown below. Figure 6 shows a comparison of some example OOD images generated by GPT-4 with original image.

```
Generate an image variant containing the creature in the
provided image.  keep the key features of this creature
unmodified.  You must show the full body view of this
creature.
```

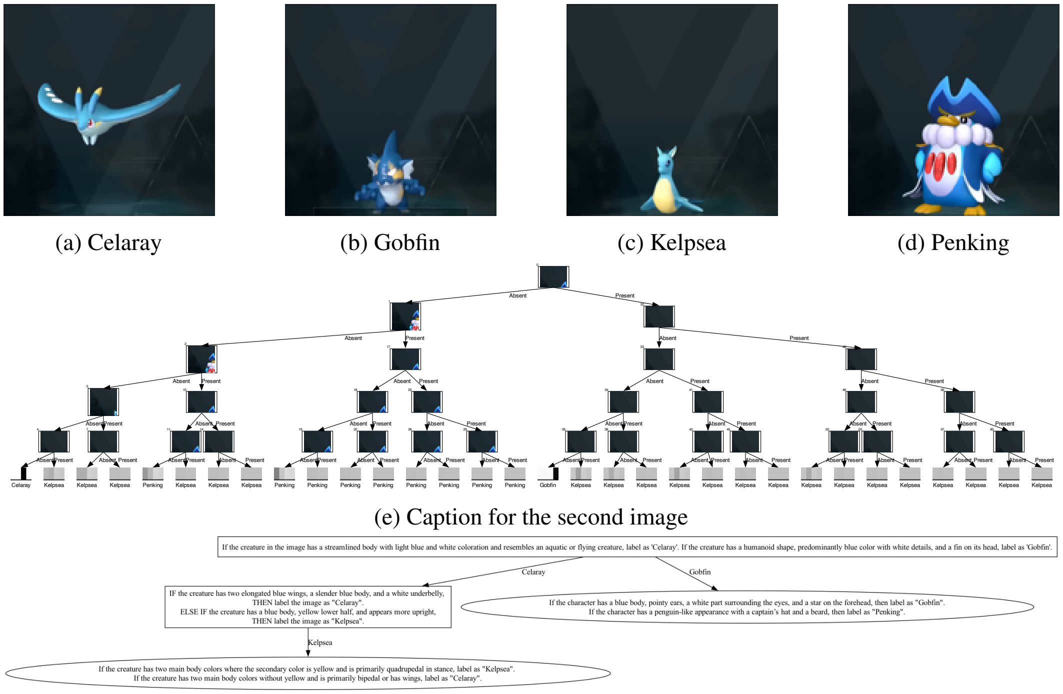
8.3.3 Human Evaluation Protocol

We conduct user study to access the interpretability of our method and ProtoTree. For both methods, we send (1) the original image datasets and (2) visualizations of the discovered programs to the human raters, and as the human rater to make predictions based on those programs. We then compute the accuracy of their predictions, and report the mean and standard deviations. We select the group of human raters so that they have no background in machine learning research.

8.4 Limitations

We acknowledge the following limitations, which merit further exploration in future studies. It is important to note that these limitations pertain to the specific, simplified instantiation of the algorithms used in this preliminary study, rather than to the LSP framework itself:

- **Domain-Specific Language Design:** A common practice in NSp is to design DSLs suitable for specific tasks. This work presents only a basic example of a DSL designed for predictive tasks. Investigating a variety of DSL designs could enable LSPs to excel across a broader range of applications.
- **Program Complexity:** Our search algorithm prioritizes accuracy without considering the complexity of the resulting programs, potentially leading to redundancies. The complexity of the learned programs could be reduced either through post-processing (akin to code cleaning) or by integrating complexity regularization during the search process.



(a) Celaray

(b) Gobfin

(c) Kelpsea

(d) Penking

(e) Caption for the second image

If the creature in the image has a streamlined body with light blue and white coloration and resembles an aquatic or flying creature, label as 'Celaray'. If the creature has a humanoid shape, predominantly blue color with white details, and a fin on its head, label as 'Gobfin'.

IF the creature has two elongated blue wings, a slender blue body, and a white underbelly,
THEN label the image as "Celaray".
ELSE IF the creature has a blue body, yellow lower half, and appears more upright,
THEN label the image as "Kelpsea".

If the character has a blue body, pointy ears, a white part surrounding the eyes, and a star on the forehead, then label as "Gobfin".
If the character has a penguin-like appearance with a captain's hat and a beard, then label as "Penking".

If the creature has two main body colors where the secondary color is yellow and is primarily quadrupedal in stance, label as "Kelpsea".
If the creature has two main body colors without yellow and is primarily bipedal or has wings, label as "Celaray".

(f) Example

Figure 7: **Example programs discovered by LSP (bottom) and ProtoTree (middle).** While ProtoTree offers some interpretability by displaying prototype image patches to the user, it can be misleading as there is no guarantee that the prototypes are meaningful (e.g. many patches miss the key regions, and there also exists entire branches that overfit to the background). In contrast, the programs discovered by LSP accurately capture the characteristics of the creatures and guide the decision-making process step by step.

8.5 Societal Impact

The development and deployment of interpretable predictive models using Large Language Models (LLMs) have significant societal implications. By enhancing the transparency and interpretability of AI systems, our approach addresses critical concerns related to trust, accountability, and fairness of the decision making process. These improvements are particularly valuable in high-stakes domains such as healthcare, finance, and legal decision-making, where understanding the rationale behind AI decisions is crucial for gaining user trust and ensuring ethical outcomes.

However, as with any AI technology, careful consideration must be given to the potential risks of misuse or unintended consequences. It is essential to continue developing comprehensive guidelines and regulatory frameworks to ensure that the deployment of these models aligns with societal values and ethical standards. By promoting transparency and interpretability, our approach paves the way for more responsible and beneficial integration of AI into society.

8.6 License

The open-source code from GitHub used in this paper adheres to various licenses like MIT, Apache 2.0, and GPL, ensuring the code's free use, modification, and distribution under specific conditions. The ChatGPT API from OpenAI and the Gemini API from Google are used in compliance with their respective terms of service, which include usage restrictions, attribution requirements, and provisions for commercial use. By following these licenses and terms, we maintain ethical and legal standards in utilizing both open-source code and proprietary APIs in our research.