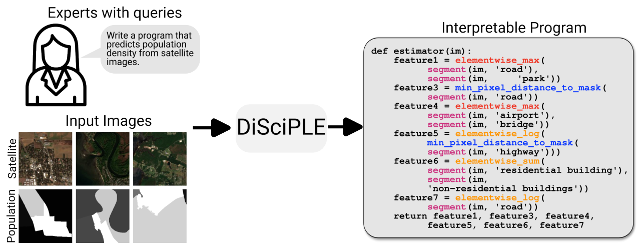
DiSciPLE: Learning Interpretable Programs for Scientific Visual Discovery

Utkarsh Mall[1] Cheng Perng Phoo[2] Mia Chiquier[1] Bharath Hariharan[2] Kavita Bala[2] Carl Vondrick[1]
[1]Columbia University [2] Cornell University
Correspondence: `um2171@columbia.edu`
disciple.cs.columbia.edu

Abstract

Visual data is used in numerous different scientific workflows ranging from remote sensing to ecology. As the amount of observation data increases, the challenge is not just to make accurate predictions but also to understand the underlying mechanisms for those predictions. Good interpretation is important in scientific workflows, as it allows for better decision-making by providing insights into the data. This paper introduces an automatic way of obtaining such interpretable-by-design models, by learning programs that interleave neural networks. We propose DiSciPLE (Discovering Scientific Programs using LLMs and Evolution) an evolutionary algorithm that leverages common sense and prior knowledge of large language models (LLMs) to create Python programs explaining visual data. Additionally, we propose two improvements: a program critic and a program simplifier to improve our method further to synthesize good programs. On three different real-world problems, DiSciPLE learns state-of-the-art programs on novel tasks with no prior literature. For example, we can learn programs with 35% lower error than the closest non-interpretable baseline for population density estimation. The supplementary material can be found at: https://disciple.cs.columbia.edu/pdf/supplementary.pdf



```
def estimator(im):
    feature1 = elementwise_max(
          segment(im, 'road'),
          segment(im,     'park'))
    feature3 = min_pixel_distance_to_mask(
          segment(im, 'road'))
    feature4 = elementwise_max(
          segment(im, 'airport'),
          segment(im, 'bridge'))
    feature5 = elementwise_log(
          min_pixel_distance_to_mask(
          segment(im, 'highway')))
    feature6 = elementwise_sum(
          segment(im, 'residential building'),
          segment(im,
          'non-residential buildings'))
    feature7 = elementwise_log(
          segment(im, 'road'))
    return feature1, feature3, feature4,
          feature5, feature6, feature7
```

Figure 1. We introduce a framework to discover interpretable, predictive programs for scientific computer vision tasks.

1. Introduction

Many modern scientific workflows are built on top of visual data. Researchers in remote sensing, climate science, ecology, and other sciences use images as a window into our world to estimate population density, the amount of biomass, poverty indicators, and so on. There is a massively increasing volume of visual data, be it from an ever-expanding set of satellites, widespread camera traps, or images uploaded on the web, that is available to domain experts, and computer vision has the potential to meaningfully assist scientists in using it for scientific insight.

Scientific applications of computer vision, however, are demanding tasks because we want models that not only predict outcomes but also reveal underlying mechanisms. For example, a researcher who studies demography may be able to train excellent predictive models that learn the relationships between a satellite image and the population. However, understanding why certain regions are densely populated is crucial for urban planning and policy decisions — black-box predictions offer no interpretation of what makes a region have a high population. Scientists themselves want to derive insight from the models, not just predict.

While there have been many works that train interpretable vision models (for example concept-bottlenecks [20, 28]), these models are often limited to simple functions of primitive concepts, such as bag of words. These simple functions do not scale to the realistic complexity of our visual world and the complex relationships between its many rich scientific indicators, resulting in poor accuracy. A promising direction to model rich relationships without foregoing interpretability is to learn *programs* on top of conceptual primitives. Recently, code generation methods such as ViperGPT and VisProg [14, 37] have demonstrated that large language models are able to synthesize programs with competitive performance on many vision tasks, and the representations are interpretable by construction because programs are human-readable. However, while these methods work well for established vision tasks, they often fail to generalize to scientific applications of computer vision because the tasks are new and outside the scope of the training data on the internet. LLMs lack the requisite knowledge to answer novel or domain-specific questions, and as we will show, directly applying such zero-shot code generation methods on scientific domains is not effective.

How can we automatically discover accurate and interpretable programs from the volumes of visual data in scientific applications? This paper introduces DiSciPLE, a framework for **Di**scovering **Sci**entific **P**rograms using **L**LMs and **E**volution. Given a large dataset of images, our approach learns to synthesize a program for solving the task. As the name suggests, DiSciPLE introduces an evolutionary search algorithm that starts with zero-shot programs from LLMs and iteratively improves them over the dataset. The discovered programs are able to interleave neural networks, in particular open-world segmentation foundation models, for segmentation with logical and mathematical operations, enabling powerful predictions while also being interpretable. Our framework makes several improvements to integrate evolutionary search with LLMs, using program simplification and critics to provide fine-grained guidance that accelerates program search.

In three different scientific applications of computer vision, DiSciPLE is able to learn state-of-the-art programs for novel tasks that have no prior documented solutions in existing literature. Our approach significantly outperforms neural networks at estimating population density from satellite imagery. Our method also obtains strong out-of-distribution performance at estimating a region's biomass, generalizing to geographical regions outside of the training set significantly better than all other baselines.

Our contributions are:

- We introduce a novel framework **DiSciPLE**, that can produce interpretable, reliable, and sample-efficient programs for scientific discovery.
- We present two key components: a critic and a program simplification method to DiSciPLE that can further improve the evolutionary search resulting in better programs.
- We propose benchmarks for the task of scientific visual discovery containing real-world high-dimensional visual data for three problems in two different domains. We also apply DiSciPLE on these benchmarks and show that our learned programs are more interpretable, reliable, and data-efficient compared to baselines.

2. Related Works

Concept bottlenecks. Concept bottleneck [20, 33, 42] is an approach used to create interpretable-yet-powerful classifiers. The key idea is to train a deep model to predict a set of low-level concepts or bottlenecks and then learn a linear classifier. Such concept bottlenecks have the basis of methods in several areas such as fine-grained recognition [11, 17, 38, 45] and zero-shot learning [1, 19, 22]. However in order to train these models, expensive data is needed to be collected for the bottleneck concepts themselves. One way to reduce this annotation cost is to sequentially ask questions in an information-theoretically optimized way [2, 3]. Researchers have also automated this pipeline by using large-language models as a knowledge base to propose concept bottleneck models [15, 28, 34]. [4] proposed an evolutionary algorithm with LLMs as the mutation operation to discover interpretable concept bottleneck models without prior information. While these models are interpretable, they are very simple in terms of expressive power. In this work, we instead evolve programs, that are more expressive than a bag of words, while being interpretable.

Symbolic regression. Symbolic regression (SR) [5] is a technique for learning equations through evolutionary search. Several methods have been proposed to improve the search efficiency [25], however, most SR techniques cannot solve problems beyond simple mathematical formulas, with simple mathematical primitives. This is partly because the search space of solutions is combinatorially too large. As a result, SR methods fail to work for images, which are too high-dimensional. Our method instead focuses on problems with high-dimensional visual data, by leveraging visual foundation models as primitives. This results in models that are better performing while being interpretable. Like our approach, recent work on SR [12, 23, 29, 35] has also looked at using LLMs to better guide the search. However, these methods are only tested on lower-dimension mathematical problems for formula discovery, with a limited set of primitives. We instead propose an approach that is complementary to these methods. Methods for SR cannot be applied directly in higher-dimensional open-world visual problems, on the other hand on low-dimensional problems existing tools for SR [12] would perform better than DiSciPLE. The focus of this work is on such real-world problems, where the primitive functions are more complex than mathematical operations and can even be open-world, for example, a text-to-image segmentation.

Neuro-Symbolic Program Learning [6, 27] is another avenue for learning programs for observation datasets or question answering. These methods typically try to learn both discrete program structures together with neural networks. However, since this optimization is non-differentiable these methods require reinforcement learning [18] or complex non-differentiable optimization techniques [10]. The hard optimization issue makes the problem of learning programs sample inefficient in real-world settings. We alternatively use LLMs ability to program to better guide the search for such programs.

Program synthesis with LLMs. Several works have utilized LLM coding ability in different applications such as VQA [13, 37] and robot manipulation [24]. While the zero-shot inferred code work very well on domains well-known to the internet, they tend to perform poorly on problems in scientific domains, as shown by our results.

Scientific applications. Researchers in numerous scientific domains have used machine learning tools to build predictive models for their quantities of interest. In this work, we focus on two such scientific domain of: demography and climate science. For both these domains, we use remote sensing vision language foundation models as powerful primitives, along with mathematical, logical and image operators. In demography, we focus on the problems of socioeconomic indicator prediction [44], namely population density and poverty estimation [31, 41]. Similarly in climate science we focus on the problem of aboveground biomass prediction (AGB) [32].

3. Methodology

Our key contribution is a program search framework that leverages LLMs to perform evolutionary search. In Sec. 3.1 we formalize the problem of program search. In Sec. 3.2 we present our method of incorporating LLMs in the evolutionary search framework. Finally, in Sec. 3.3, 3.4 and 3.5, we discuss the improvements to this framework to speed-up the search.

3.1. Problem Formulation

Our system receives as input a **dataset** $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ consisting of inputs $x_i \in X$ and quantities of interest $y_i \in Y$. For example, when estimating geospatial indicators like poverty, x_i may be a latitude and longitude, along with other metadata about that location. The system must produce an *interpretable program* $P : X \rightarrow Y$ that maps inputs to corresponding outputs. As is typical with standard supervised learning, we also assume a loss function or **metric** $\mathcal{M}$ that measures how good a particular prediction is, and seek a program that yields the best evaluation score:

$$s(P; \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{M}(P(x_i), y_i) \tag{1}$$

For example, in the case of population density a good metric used by domain experts is L2 error over log *i.e.* $\mathcal{M}(y', y) = ||log(y') - log(y)||_2$ [30, 31].

For our programs to be interpretable, they must put together modules or **primitives** in an interpretable way. We conceptualize these primitives as a library of functions $\mathcal{F} = \{f_1, f_2 \ldots f_k\}$, that can be used to construct a program P. Given that we are analyzing visual data, a key primitive will be an open-vocabulary recognition model that is applied to any imagery associated with a data point. For example, when estimating poverty for a location, we can define a primitive that queries the satellite images available at that location and uses an open-vocabulary recognition engine such as GRAFT [26] to detect/segment various concepts. Recent advances in recognition have produced such foundation models for a range of modalities [26, 36]. In addition, we will assume basic mathematical, logical and image operators such as logarithms, elementwise maximum, or a distance transform. We note that the set of primitives is often not specific to the task and can be shared across a range of problems in a domain. That said, the set of primitives can be expanded upon if needed by domain experts for particular problems; for example, a climate scientist may want to include a function that can look up the average temperature at a particular place and time.

Finally, to enable us to search the space of programs effectively and leverage the conceptual understanding of LLMs, we assume that we have a natural language name or description $descr$ of the quantity of interest Y. For example, this may be the phrase "population density". As we will see below, this information will be useful in guiding the LLM to search the space well.

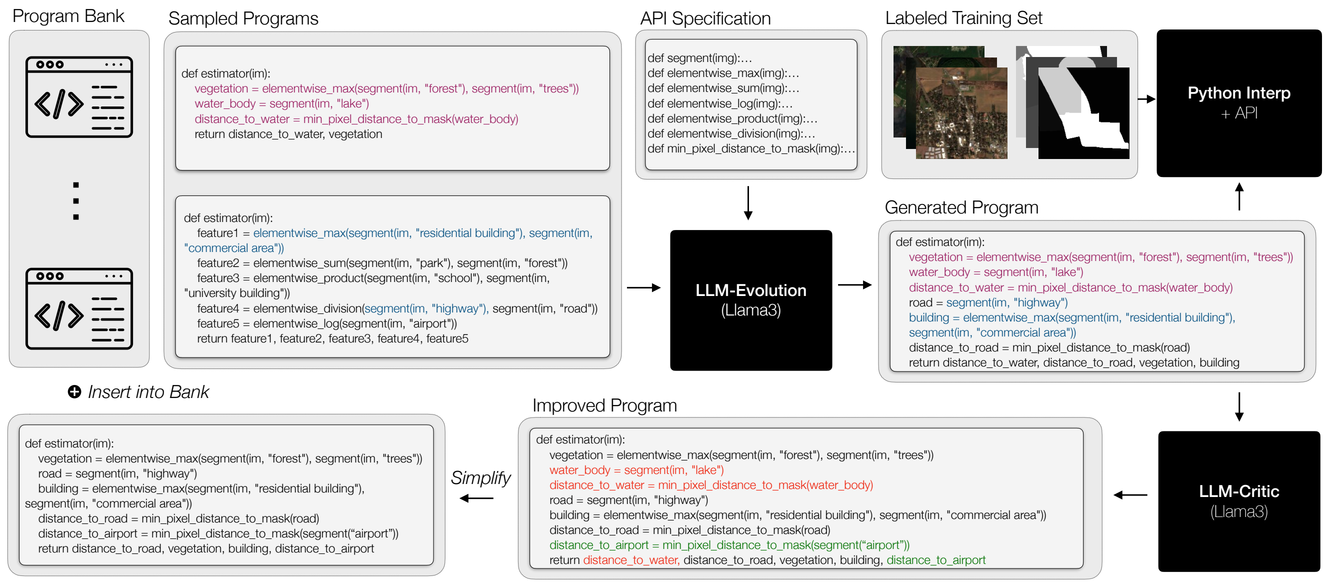
Putting everything together, our proposed system, DiSciPLE (**Di**scovering **Sci**entific **P**rograms using **L**LMs and **E**volution), takes as input the dataset $\mathcal{D}$, the metric $\mathcal{M}$, the set of primitives $\mathcal{F}$ and the textual description $descr$. It produces an interpretable and accurate program P that maps inputs X to the output quantity of interest Y.

We next describe our proposed system.

3.2. Evolutionary Search for Programs

To search through the vast, discrete space of programs, DiSciPLE adapts evolutionary search. Evolutionary program search typically starts with a large population of random programs. These programs are then sampled based on their fitness as parents. The parent programs create new programs through crossover and mutation, resulting in a new population. Newer generations improve over the previous as the population is getting optimized for the fitness function. We use the metric $\mathcal{M}$ as the fitness function in our work.

We keep the overall evolutionary algorithm the same but replace key steps with an LLM. First, at the start of the process, we provide the LLM with a prompt for the objective to generate the initial programs. To leverage the prior knowledge of LLMs, we use a prompt p_o that mentions the specified description of the quantity of interest: "*Given a satellite image, write a function to estimate* $\langle descr \rangle$". We do not expect the LLM to answer such a difficult scientific question without leveraging the observations $\mathcal{D}$; however, the prompt prevents the evolutionary algorithm from searching in com-



```
def estimator(im):
    vegetation = elementwise_max(segment(im, "forest"), segment(im, "trees"))
    water_body = segment(im, "lake")
    distance_to_water = min_pixel_distance_to_mask(water_body)
    return distance_to_water, vegetation
```

```
def estimator(im):
    feature1 = elementwise_max(segment(im, "residential building"), segment(im,
"commercial area"))
    feature2 = elementwise_sum(segment(im, "park"), segment(im, "forest"))
    feature3 = elementwise_product(segment(im, "school"), segment(im,
"university building"))
    feature4 = elementwise_division(segment(im, "highway"), segment(im, "road"))
    feature5 = elementwise_log(segment(im, "airport"))
    return feature1, feature2, feature3, feature4, feature5
```

```
def segment(img):…
def elementwise_max(img):…
def elementwise_sum(img):…
def elementwise_log(img):…
def elementwise_product(img):…
def elementwise_division(img):…
def min_pixel_distance_to_mask(img):…
```

```
def estimator(im):
    vegetation = elementwise_max(segment(im, "forest"), segment(im, "trees"))
    water_body = segment(im, "lake")
    distance_to_water = min_pixel_distance_to_mask(water_body)
    road = segment(im, "highway")
    building = elementwise_max(segment(im, "residential building"),
    segment(im, "commercial area"))
    distance_to_road = min_pixel_distance_to_mask(road)
    return distance_to_water, distance_to_road, vegetation, building
```

```
def estimator(im):
    vegetation = elementwise_max(segment(im, "forest"), segment(im, "trees"))
    water_body = segment(im, "lake")
    distance_to_water = min_pixel_distance_to_mask(water_body)
    road = segment(im, "highway")
    building = elementwise_max(segment(im, "residential building"), segment(im, "commercial area"))
    distance_to_road = min_pixel_distance_to_mask(road)
    distance_to_airport = min_pixel_distance_to_mask(segment("airport"))
    return distance_to_water, distance_to_road, vegetation, building, distance_to_airport
```

```
def estimator(im):
    vegetation = elementwise_max(segment(im, "forest"), segment(im, "trees"))
    road = segment(im, "highway")
    building = elementwise_max(segment(im, "residential building"),
segment(im, "commercial area"))
    distance_to_road = min_pixel_distance_to_mask(road)
    distance_to_airport = min_pixel_distance_to_mask(segment("airport"))
    return distance_to_road, vegetation, building, distance_to_airport
```

Figure 2. Overview of our evolutionary algorithm with *critic* and *simplification*. We start with an initialized bank of program trying to solve a task. From this bank we sample pairs of programs based on their fitness score and perform crossover/mutations over them to produce new programs. The generated program is further improved by passing it through a critic and then an analytical simplification step. This program is then evaluated and put in the next generation of program bank. The evaluation score of the program is used to determine the fitness for the next generation of evolution.

pletely random directions. As a result, our initial population is not entirely random.

Second, rather than using the symbolic methods of crossover and mutation, we use the LLM to perform these operations. LLMs have common sense about programming and result in much better program modifications when performing crossover and mutations. More specifically, let $P^t_{k_1}$ and $P^t_{k_2}$ be two programs sampled from the t^{th} generation selected as parents based on the fitness function. To perform a crossover operation we pass, the objective prompt p_o, the two programs $P^t_{k_1}$ and $P^t_{k_2}$, their corresponding scores (using Eq. (1)), along with a crossover prompt p_c to obtain a new program:

$$P^{t+1}_k = \mathcal{LLM}(P^t_{k_1}, P^t_{k_2}, s(P^t_{k_1}; \mathcal{D}), s(P^t_{k_2}; \mathcal{D}), p_o, p_c) \tag{2}$$

The crossover prompt instructs the LLM to make use of the two-parent program and come up with a new program. The LLM is able to combine elements from the parents to produce something new as can be seen in Fig. 2. Please refer to the supplementary for more examples.

Similar to crossover, we also mutate a program with some probability using a mutation prompt p_m.

$${}^{m}P^{t+1}_k = \mathcal{LLM}(P^{t+1}_k, s(P^{t+1}_k; \mathcal{D}), p_o, p_m) \tag{3}$$

We present the exact input fed to the LLM for crossover and mutation in the supplementary. Note that both crossover and mutation operations also include the objective prompt preventing the LLM from generating programs far away from the objective.

3.3. Feature Set Prediction

Solutions for many problems require combination of multiple feature. Optimizing for the correct combination of these features is challenging for an LLM, as we are not doing gradient-based optimization. Therefore instead of prompting the LLM to directly generate a predictor for y, we prompt it to create a list of predictive features. We then learn a linear regressor on top of this list and use the regressor with the list of features as the final program. Since both the generated program and regressor are interpretable, our final program is also interpretable.

3.4. Program Critic

The only form of supervision our method gets is through the metric score $s(P; \mathcal{D})$ created by evaluating the program P on the observations. However, since we perform crossover and mutation through a language model, we can provide finer-grained information to LLM in order to aid the search.

More specifically, we propose a critic that performs a finer-grained evaluation of the program that we get after crossover/mutation. In most visual domains, the data can categorically distributed by using the same set of foundational primitives. Our critic performs a *stratified evaluation* by partitioning the observation data into multiple categories

Algorithm 1: DiSciPLE's learning loop

```
Input: Observation set
       $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$,
       metric $\mathcal{M}$, an objective prompt $p_o$, a set of
       primitive function $\mathcal{F} = \{f_1, f_2 \ldots f_k\}$
Hyperparams: Mutation probability $\rho_m$, total
       number of generations $T$,
       population size $M$, crossover $p_c$
       and and mutation $p_m$ prompts.
Output: A program $P^*$ in the form of a program
       that explains the observations best.
1  $B^0 \leftarrow \{\}$ // Initialize a programs bank
2  for $i = 1, \ldots, M$ do
3      $P_i^0 \leftarrow \mathcal{LLM}(p_o)$
4      $B^0 \leftarrow B^0 \cup \{P_i^0\}$
5  $P^* \leftarrow P_i^0$
   // Evolution loop
6  for $t = 0, \ldots, T$ do
7      $B^{t+1} \leftarrow \{\}$
8      for $i = 1, \ldots, M$ do
9          $P_{k_1}^t, P_{k_2}^t \leftarrow$ sample_parents$(B_t)$ // Sample
               parents for crossover
10         $P_i^{t+1} \leftarrow$
               $\mathcal{LLM}(P_{k_1}^t, P_{k_2}^t, s(P_{k_1}^t; \mathcal{D}), s(P_{k_2}^t; \mathcal{D}), p_o, p_c)$
               // crossover operation
11         if $u \sim \mathcal{U}(0,1) < \rho_m$ then
12             $P_i^{t+1} \leftarrow$
                   $\mathcal{LLM}(P_k^{t+1}, s(P_k^{t+1}; \mathcal{D}), p_o, p_m)$
                   // mutation operation
           // critique and simplification
13         $P_i^{t+1} \leftarrow$critic$(P_i^{t+1})$
           $P_i^{t+1} \leftarrow$simplifier$(P_i^{t+1})$
14         if $s(P_i^{t+1}; \mathcal{D}) > s(P^*; \mathcal{D})$ then
15             $P^* \leftarrow P_i^{t+1}$
16         $B^{t+1} \leftarrow B^{t+1} \cup \{P_i^{t+1}\}$
17 return $P^*$;
```

and evaluating the program on individual strata.

$$\mathcal{D} = d_1 \cup d_2 \cup \cdots \cup d_c, \quad \text{where } d_i \cap d_j = \emptyset \text{ for } i \neq j \quad (4)$$

Since all the problems in our benchmark are geospatial, we use a critic that takes the satellite image corresponding to each input and uses a segmentation model to partition the observation dataset into land-use categories. The critic obtains per-partition score $s(P; d_i)$ and prompts the LLM to improve the program on categories the model is bad. The addition of a critic improves the programs on data overlooked by existing programs, resulting in reliable programs.

3.5. Program Simplification

Successive steps of crossovers and mutations of programs result in large programs with many redundancies, hurting interpretability. We propose an analytical approach to simplify the programs and remove the redundant parts of it. Our generated programs can be represented as a directed acyclic graphs (DAG) (we use the abstract syntax tree (AST); see supplementary). In these DAGs, all the constants and the arguments of the function are root nodes. Only the return statement and the unused variables are the leaf nodes. Any leaf node that is not a return statement, is a piece of code that is not needed and can be removed. We then recursively remove all the leaf nodes that are not return statements.

While removing such nodes (unreachable by the return statement) is useful, there could still be features that are returned by the program but are not contributing. Recall that we use linear regression on the list of features returned by the program. The weights assigned to individual features by the regression model are useful indicators of which features are redundant. We remove the features that have a significantly smaller weight compared to the largest weight in the regression (a threshold of 5% works well). Removing these features from the return statement results in several newly created leaf nodes. We therefore, redo the recursive leaf node removal to further simplify the program. Each generated program is first improved through the critic and then simplified before adding back to the population. Algorithm 1 shows the complete process.

4. Results

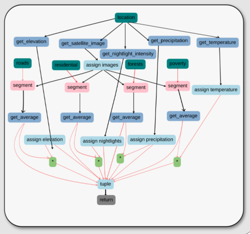
4.1. Implementation details

For all our main experiments we used an open-source LLM *llama-3-8b-instruct* [9] served using the vLLM library [21]. However, in the supplementary, we also explore other open-source language models. All the visual data in our benchmarks comes from satellite images, so to allow inferring semantic information from it, we use a black-box open-world foundational model for satellite images, GRAFT [26]. Some experiments use ground-truth annotations from OpenStreetMaps [40] as an alternative to disentangle the effect of segmentation from discovery.

We run our evolutionary method for $T = 15$ generations with a population size of $M = 100$. For all the problems, the input observation data comes from different geographical locations around the world. We split this data into three parts. Two-thirds of the easternmost observations are used to create a training-testing split. The remaining one-third of the data is use to evaluate reliability (out-of-distribution generalization). We also release this benchmark for future research in this area.

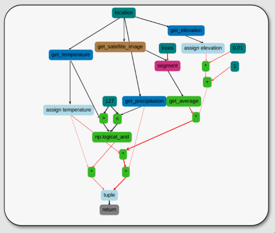
Poverty

```
def estimator(location):
    images = get_satellite_image(location)
    roads = segment(images, 'roads')
    buildings = segment(images,
         'residential building')
    forests = segment(images, 'forests')
    avg_roads = get_average(roads)
    avg_buildings = get_average(buildings)
    avg_forests = get_average(forests)
    poverty_mask = segment(images, 'poverty')
    avg_poverty = get_average(poverty_mask)
    temperature = get_temperature(location)
    elevation = get_elevation(location)
    nightlights = get_nightlight_intensity(location)
    precipitation = get_precipitation(location)
    return (avg_poverty, avg_roads, avg_buildings,
        avg_forests, elevation, nightlights,
        precipitation, avg_poverty * temperature,
        avg_roads * elevation, avg_buildings *
        nightlights, avg_forests * precipitation)
```



AGB

```
def estimator(location):
    images = get_satellite_image(location)
    smalltrees = segment(images, 'trees')
    avg_biomass = np.logical_and(
        get_precipitation(location) > 127,
        get_temperature(location) > 127)
    avg_trees = get_average(smalltrees)
    elevation = get_elevation(location)
    temperature = get_temperature(location)
    precipitation = get_precipitation(location)
    feature1 = avg_biomass * (temperature)
    feature2 = avg_trees *
        (1 + 0.01 * elevation) *
        (temperature) * (precipitation)
    return feature1, feature2
```



Population Density

```
def estimator(im):
    feature1 = elementwise_max(
          segment(im, 'road'),
          segment(im,     'park'))
    feature3 = min_pixel_distance_to_mask(
          segment(im, 'road'))
    feature4 = elementwise_max(
          segment(im, 'airport'),
          segment(im, 'bridge'))
    feature5 = elementwise_log(
          min_pixel_distance_to_mask(
          segment(im, 'highway')))
    feature6 = elementwise_sum(
          segment(im, 'residential building'),
          segment(im,
          'non-residential buildings'))
    feature7 = elementwise_log(
          segment(im, 'road'))
    return feature1, feature3, feature4,
          feature5, feature6, feature7
```

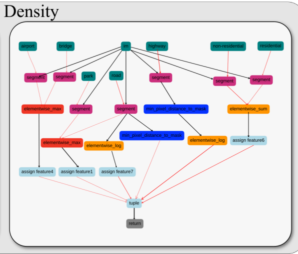


Figure 3. The best performing programs for each of the 3 benchmark problems as Python programs (left in each card) and the corresponding DAG representation on the right. The DAG representation allows better visualization of the importance of different components. The thickness of the red edges determine how important that component is. A black edge represents computation; when removed it is either the same as one of its subsequent edges or removing it could result in a bug.

4.2. Benchmark for Visual Program Discovery for Scientific Applications

Given the novelty of the visual program discovery task, there exists no pre-existing benchmark. We define a new benchmark for this task, drawing on scientifically relevant geospatial problems. Concretely, we choose two different problems in *Demography*: population density and poverty indicators, and to a problem in *Climate Science*: for above ground biomass (AGB) estimation.

It is important to note that for these problems, *true relationships between variables of interest are actually unknown*. As such, an LLM cannot be expected to produce a good program in a zero-shot manner, because it has never seen these relationships before. This is in contrast to problems like VQA [37] where the reasoning required to answer a question is well known and we can simply rely on the LLM's world knowledge. In the case of scientific discovery, actual data is needed to discover the right reasoning.

In the following, we present the observation datasets, metrics, and overview of primitives.

4.2.1. Population Density

Observation Dataset: The problem seeks to predict the population density by observing the satellite images of a region [30, 31]. We obtain the population density values (y_i) for various locations in the USA by using ACS Community Surveys 5-year estimates [39]. Input observations (x_i) are sentinel-2 satellite images at a resolution of 10m [7]. For this experiment, we also use OpenStreetMaps masks [40] for 42 different land-use concepts (see supplementary) as part of the input.

Metric and Primitives: Population density values are aggregated at the county block group level. The predicted population densities are therefore also aggregated at the county block group level. The metric is the per-block group level average L2 error after applying a log transformation. Along with the arithmetic, and logical primitives (see supplementary) , we use open-vocabulary segmentation as a primitive. The segmentation function returns a binary mask for an input concept.

4.2.2. Poverty Indicator

Observation Dataset: For poverty estimation, we use data from SustainBench [43]. The dataset contains coordinate location as input and wealth asset index as output.

Metric and Primitives: We use L2 error for each location as the evaluation metric. To obtain semantic land use information about a location, we first define a *get_satellite_image* function, that returns a sentinel-2 satellite image for any location. This can be used in conjunction with the open-world satellite image recognition model to obtain semantic information about the world. Other than this we also include as primitives functions that return average annual temperature, precipitation, nightlight intensity, and elevation at the input location.

4.2.3. Aboveground Biomass

Observation Dataset: Similar to poverty estimation, the observation variables are an input location and the output AGB estimate. We use NASA's GEDI [8] to obtain the observation value for three US states. We use data from Massachusetts and Maine (North-East) as the train/test set and Washington (NorthWest) as the out-of-distibution set.

Metric and Primitives: We use L2 error as the metric and the same primitives as poverty estimation.

4.3. Experimental Setup

For the same set of training data we compare our best generated program with a set of baselines.

1. **Mean:** A naive baseline that use the mean of the training observation as the prediction.

2. **Concept Bottleneck (CB):** Similar to [20, 33, 42], we first extract a list of relevant features and train a linear classifier on it. This method is interpretable due to the bottleneck, however it is not very expressive (see supplementary).
3. **Deep models:** We use deep models such as ResNets [16] as baseline (see supplementary for details). We use a small and large variant for each.
4. **Zero-shot:** This baseline tests how good would LLMs be on their own in generating programs solely relying on prior knowledge without any observation. Since the generated programs can vary drastically, we report an average of 5 different zero-shot programs.
5. **Random Search:** Instead of evolutionary search, this baseline relies on the stochasticity of LLMs to perform a random search. If DiSciPLE is better at searching, it should do better than random searching for the same number of calls to an LLM.

4.4. Results and Discussion

We first test our programs on unseen *in-domain* observations close to the regions used for training (Tab. 1 (left)). We observe that DiSciPLE outperforms all interpretable baselines. It can even outperform a deep model in many cases, specifically on population density estimation, while being significantly more interpretable. DiSciPLE also outperforms zero-shot program inference from LLMs. As discussed before, this is in line with the fact that DiSciPLEis uncovering new relationships that may not be known to us, and by extension, to the LLM. The performance of random search while better than zero-shot is significantly worse than DiSciPLE. This shows that DiSciPLE is able to perform a significantly faster search, by reducing the meaningful search space. Our evolutionary process effectively leverages data to perform this novel discovery.

Are our programs reliable? If a program is reliable it should be able to generalize to other regions. Tab. 1 (right) shows DiSciPLE to these baselines on such an out-of-distribution set. Here our approach outperforms all baselines *including deep networks*, suggesting that due to its interpretable-by-design representation, our method learns a model that can generalize better and overfit less to the in-distribution training data.

We also show these results qualitatively in Fig. 4, by comparing population density predictions of DiSciPLE and the baselines to the true population density. It is very clearly evident that DiSciPLE can model the fine-grained changes in population in unseen regions significantly better than the baselines (refer to supplementary for more visualizations).

Are our programs data-efficient? Our methods are only trained on a maximum of 4000 observations. Fig. 5 further shows that even when the amount of training data is reduced, our approach shows minimal degradation in performance compared to deep networks. This suggests that while deep models can learn to generalize with a lot more data, our model does not need as much data to begin with, making it data-efficient.

Are our programs interpretable? Our programs are interpretable-by-design as we can visualize the factors contributing to performance. Fig. 3 shows such programs (left in each card) for all the problems in our benchmark. An expert who is working with our method to figure out such programs can add/edit parts of the formula and figure out which/how much do each of these components matters.

We perform this step of understanding the influence of individual operations by removing each operation in our program and measuring its effects on the final score. The DAGs on the right of each program show the program structure and the red edges show the influence of each component proportional to the width. This visualization can allow experts to understand which operations are important for the model. For example, in the program for population density Fig. 3, we can see that semantic concepts such as “highway” and “residential building” are very important.

Can our method perform better than expert humans? Our method would only be useful in real-world scenarios if it can come up with stronger or comparable programs to human experts. We test this on the task of AGB, by providing an expert (a PhD student actively working on AGB) with a user interface with the same information as our method. The experts took about 1.5 hours to use their domain knowledge and iterate over their program for AGB estimation. However, the best program they could come up with had an L1 error of **37.65** on the in-distribution set and **53.20** on the OOD set (compared to **24.79** and **31.10** for DiSciPLE). We figure this is primarily because experts need to spend more time on the problem. In general, experts would spend numerous days to come up with a good program, while our method can come up with a better program faster.

Extension to more indicators We also test DiSciPLE on a larger suite of demographic indicators. Using SocialExplorer, we build a suite of 34 demography indicators. Refer to the supplementary for a list of these indicators. This includes demography information such as age group, education status, etc. In Tab. 2, we report the average performance of our method compared to baselines on this data. Since different indicators can have different scales, we first normalize all of them to have zero mean and unit standard deviation. These indicators are challenging to predict directly

Table 1. Performance of our programs on in-distribution (left) and out-of-distribution (right) observations across various problems in the proposed benchmark. This shows the reliability of programs produced by DiSciPLE (**red** is best and *blue* is second best).

	In distribution						OOD					
	Population Density		Poverty		AGB		Population Density		Poverty		AGB	
	L2-Log	L1-Log	L1	RMSE	L1	RMSE	L2-Log	L1-Log	L1	RMSE	L1	RMSE
Mean	0.6696	0.6540	1.613	1.836	42.15	50.65	0.6734	0.6561	1.591	1.844	74.15	83.02
CB	0.8298	0.7279	1.229	*1.476*	26.33	33.49	0.7951	0.7112	1.257	1.504	44.19	63.52
Deep - Small	0.4431	0.5006	1.238	1.637	30.72	37.03	0.6623	0.5967	*1.284*	*1.654*	*35.27*	*53.06*
Deep - Large	*0.3974*	*0.4843*	*1.170*	1.478	**21.15**	**27.86**	*0.4460*	*0.5115*	1.344	1.741	35.41	70.30
Zero-shot	0.4702	0.5371	1.525	1.754	38.80	46.41	0.7020	0.6412	1.510	1.773	55.11	64.32
Random Search	0.4353	0.5118	1.277	1.679	29.40	36.70	0.6763	0.6298	1.418	1.840	42.32	52.53
Ours	**0.2607**	**0.3778**	**1.077**	**1.314**	*24.79*	*32.99*	**0.3807**	**0.4426**	**1.134**	**1.420**	**31.10**	**42.93**

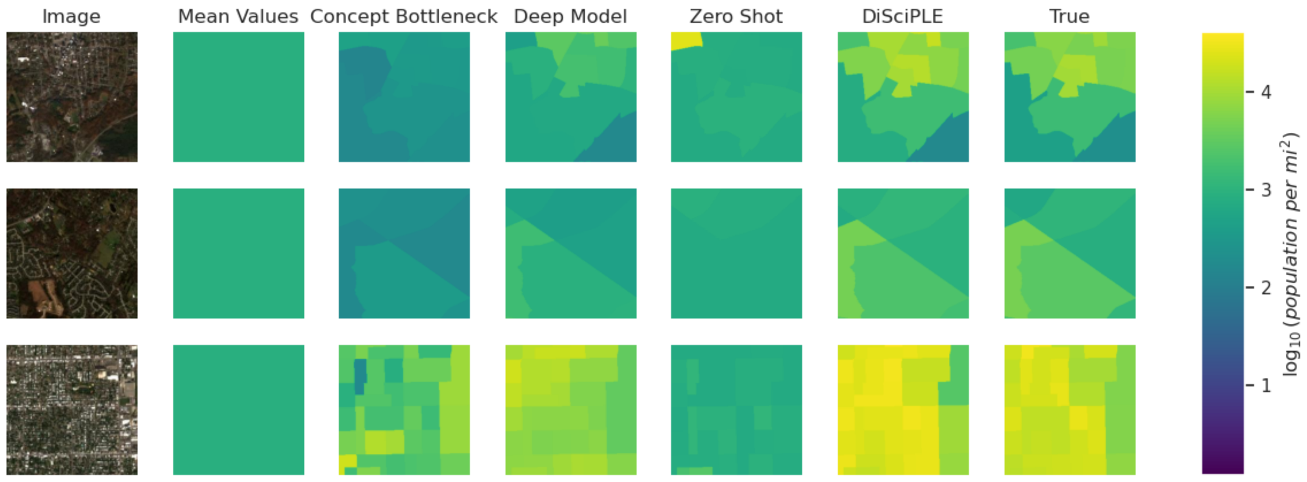


Figure 4. Qualitative comparison of DiSciPLE with other baselines on the tasks of population density. DiSciPLE Can map to the true population density maps much more accurately than the baselines (Refer to the supplementary for more comparisons). The maps display population density as the base-10 log of people per square mile.

from satellite images, as evidenced by the deep model failing to perform significantly better than CB and mean baselines. As a result while DiSciPLE performs better than all the baselines the improvements are not huge. Nonetheless, DiSciPLE performs better than every baseline. This large-scale experiment shows the potential of applying DiSciPLE to a wider range of problems. More details about these demographic indicators and individual performance on these is shown in the supplementary.

Table 2. Performance of DiSciPLE compared to baselines on a larger suite of challenging 34 demographic indicators. Since the dataset is very challenging, the deep baseline regresses to mean, however with DiSciPLE we can still see some improvements.

	Test		OOD	
	L1	RMSE	L1	RMSE
Mean	0.8578	1.1519	0.8939	1.1948
CB	0.8249	1.1159	0.8771	1.1767
Deep	0.8527	1.1556	0.8942	1.1990
Ours	**0.8159**	**1.1065**	**0.8750**	**1.1719**

4.5. Ablations

How important is the role of feature-set prediction, critic, and simplification? Table 3 measures the performance of our model on the task of population density as we successively add these components to the evolutionary algorithm. The addition of feature set prediction instead of a single feature helps, as it allows our method to learn expressive linear regression parameters instead of letting the LLM come up with them. Further adding critic results in further improvement as the programs start covering nicher concepts resulting in better unseen and OOD generalization. Finally adding in simplification also improves the program. We posit that simplification removes irrelevant features preventing the LLM from focusing on them when performing crossovers.

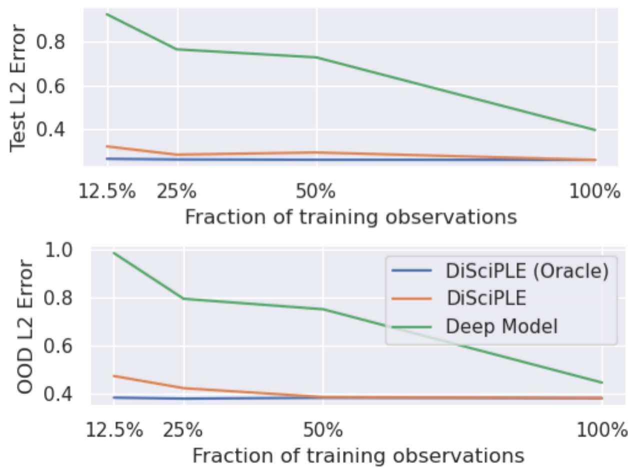


Figure 5. Performance of DiSciPLE compared to deep baselines as we reduce the amount of training observation (in terms of L2 error). The Oracle (blue) uses a program learned from all observations but uses only partial observation for parameter training. DiSciPLE (orange) uses partial observation during evolution as well. While the errors get worse as we reduce the observation data, the drop is significantly less severe for DiSciPLE compared to deep models, which tend to overfit.

Table 3. Performance of our method as we successively remove the components. Both critic and simplification lead to performance improvement for our method.

			Test		OOD	
Set	Critic	Simpli.	L2 log	L1 log	L2 log	L1 log
✗	✗	✗	0.3159	0.4296	0.4835	0.5178
✓	✗	✗	0.2906	0.4049	0.4258	0.4826
✓	✓	✗	0.2873	0.3984	0.4184	0.4684
✓	✓	✓	**0.2607**	**0.3778**	**0.3807**	**0.4426**

How important are common sense and prior knowledge of LLMs? The two major advantages an LLM provides over traditional tree-search are: 1) better crossover and mutation as LLMs can understand the meaning of the primitives. 2) use of prior knowledge for better-guided search. Therefore we remove these two sources of information and test how well can our method perform. To remove the understanding of functions we rename them with meaningless terms and remove the descriptions. To remove the context of the problem we remove the objective prompt. Tab. 4, show the performance of our method on density estimation after removing each of these prompts. Without common sense, the search cannot even progress away from the initial random programs, resulting in worse-than-mean results (L1 error of 0.84 vs 0.26 for DiSciPLE). This suggests that symbolic regression models, that have no understanding of open-world primitives, would struggle to search. If we just remove the context of the problem, the model does slightly better and can obtain results better than the mean and zero-shot programs (L1 error of 0.45). This suggests that while the search is moving in the objective's direction, it is slow.

Table 4. Perfomance of our method when removing the context of the problem (objective prompt from the evolution, and when renaming and not describing the primitive functions to the LLM. We see significant drops in performance in both cases, suggesting that both common sense and prior knowledge of LLM are important to perform efficient evolutionary search.)

Method	L1 log	L2 log
No common-sense	0.8401	0.7186
No problem context	0.4498	0.5140
DiSciPLE full	**0.2607**	**0.3778**

5. Discussion and Conclusion

Limitations: One of our limitation is that we can only differentiably optimize learnable parameters in the last computational layer. This could miss out on programs with useful parameters in some intermediate computation layers. We attempted to make the whole pipeline differentiable, however the model performance did not improve much. Many of the operations in our pipeline even though differentiable have zero-gradient in large part of input space, making gradient optimization challenging. Moreover, a completely differentiable programs is even slower to optimize resulting in much slower evolution. In future work, we plan to use initialization tricks for non-linear optimization and second-order optimization to obtain even more expressive models.

Conclusion: We present DiSciPLE – an evolutionary algorithm that leverages the prior-knowledge and common sense abilities of LLMs to create *interpretable, reliable and data-efficient* programs for real-world scientific visual data. This allows us to create programs that are more powerful than existing interpretable counterparts and more insightful than deeper uninterpretable models. We shows its prowess on 3 scientific applications by proposing a benchmark for visual program discovery. We believe that using DiSciPLE in tandem with a human expert can rapidly speed up the scientific process and result in numerous novel discoveries.

Acknowledgements This research is based upon work supported in part by the Office of the Director of National Intelligence (Intelligence Advanced Research Projects Activity) via 2021-20111000006, the NSF STC for Learning the Earth with Artificial Intelligence and Physics, the U.S. DARPA ECOLE Program No. #HR00112390060. and NSF grants 2403016 and 2403015. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, DARPA, or the US Government. The US Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

[1] Zeynep Akata, Scott Reed, Daniel Walter, Honglak Lee, and Bernt Schiele. Evaluation of output embeddings for fine-grained image classification. In *CVPR*, 2015. 2

[2] Aditya Chattopadhyay, Kwan Ho Ryan Chan, and Rene Vidal. Bootstrapping variational information pursuit with large language and vision models for interpretable image classification. In *The Twelfth International Conference on Learning Representations*, 2024. 2

[3] Aditya Chattopadhyay, Ryan Pilgrim, and Rene Vidal. Information maximization perspective of orthogonal matching pursuit with applications to explainable ai. *Advances in Neural Information Processing Systems*, 36, 2024. 2

[4] Mia Chiquier, Utkarsh Mall, and Carl Vondrick. Evolving interpretable visual classifiers with large language models. *ECCV*, 2024. 2

[5] Miles Cranmer. Interpretable machine learning for science with pysr and symbolicregression. jl. *arXiv preprint arXiv:2305.01582*, 2023. 2

[6] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural logic machines. In *International Conference on Learning Representations*, 2019. 2

[7] Matthias Drusch, Umberto Del Bello, Sébastien Carlier, Olivier Colin, Veronica Fernandez, Ferran Gascon, Bianca Hoersch, Claudia Isola, Paolo Laberinti, Philippe Martimort, et al. Sentinel-2: Esa's optical high-resolution mission for gmes operational services. *Remote sensing of Environment*, 120:25–36, 2012. 6

[8] Ralph Dubayah, James Bryan Blair, Scott Goetz, Lola Fatoyinbo, Matthew Hansen, Sean Healey, Michelle Hofton, George Hurtt, James Kellner, Scott Luthcke, et al. The global ecosystem dynamics investigation: High-resolution laser ranging of the earth's forests and topography. *Science of remote sensing*, 1:100002, 2020. 6

[9] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 5

[10] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd acm sigplan international conference on programming language design and implementation*, pages 835–850, 2021. 2

[11] Vittorio Ferrari and Andrew Zisserman. Learning visual attributes. *Advances in neural information processing systems*, 20, 2007. 2

[12] Arya Grayeli, Atharva Sehgal, Omar Costilla-Reyes, Miles Cranmer, and Swarat Chaudhuri. Symbolic regression with a learned concept library. *arXiv preprint arXiv:2409.09359*, 2024. 2

[13] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. 2023 ieee. In *CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14953–14962, 2022. 3

[14] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962, 2023. 1

[15] Songhao Han, Le Zhuo, Yue Liao, and Si Liu. Llms as visual explainers: Advancing image classification with evolving visual descriptions. *arXiv preprint arXiv:2311.11904*, 2023. 2

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 7

[17] Shaoli Huang, Zhe Xu, Dacheng Tao, and Ya Zhang. Part-stacked cnn for fine-grained visual categorization. In *CVPR*, 2016. 2

[18] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In *Proceedings of the IEEE international conference on computer vision*, pages 2989–2998, 2017. 2

[19] Elyor Kodirov, Tao Xiang, and Shaogang Gong. Semantic autoencoder for zero-shot learning. In *CVPR*, 2017. 2

[20] Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *ICML*, 2020. 1, 2, 7

[21] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023. 5

[22] Christoph H Lampert, Hannes Nickisch, and Stefan Harmeling. Attribute-based classification for zero-shot visual object categorization. *CoRR*, 2013. 2

[23] Michael Y Li, Emily B Fox, and Noah D Goodman. Automated statistical model discovery with language models. *arXiv preprint arXiv:2402.17879*, 2024. 2

[24] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023. 3

[25] Nour Makke and Sanjay Chawla. Interpretable scientific discovery with symbolic regression: a review. *Artificial Intelligence Review*, 57(1):2, 2024. 2

[26] Utkarsh Mall, Cheng Perng Phoo, Meilin Kelsey Liu, Carl Vondrick, Bharath Hariharan, and Kavita Bala. Remote sensing vision-language foundation models without annotations via ground remote alignment. *arXiv preprint arXiv:2312.06960*, 2023. 3, 5

[27] 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. *arXiv preprint arXiv:1904.12584*, 2019. 2

[28] Sachit Menon and Carl Vondrick. Visual classification via description from large language models. *International Conference on Learning Representations*, 2023. 1, 2

[29] Matteo Merler, Katsiaryna Haitsiukevich, Nicola Dainese, and Pekka Marttinen. In-context symbolic regression: Leveraging large language models for function discovery. In *ACL*, 2024. 2

[30] Nando Metzger, John E Vargas-Muñoz, Rodrigo C Daudt, Benjamin Kellenberger, Thao Ton-That Whelan, Ferda Ofli, Muhammad Imran, Konrad Schindler, and Devis Tuia. Fine-grained population mapping from coarse census counts and open geodata. *Scientific Reports*, 12(1):20085, 2022. 3, 6

[31] Nando Metzger, Rodrigo Caye Daudt, Devis Tuia, and Konrad Schindler. High-resolution population maps derived from sentinel-1 and sentinel-2. *Remote Sensing of Environment*, 314:114383, 2024. 3, 6

[32] Juan Nathaniel, Gabrielle Nyirjesy, Campbell D Watson, Conrad M Albrecht, and Levente J Klein. Above ground carbon biomass estimate with physics-informed deep network. In *IGARSS 2023-2023 IEEE International Geoscience and Remote Sensing Symposium*, pages 1297–1300. IEEE, 2023. 3

[33] Tuomas Oikarinen, Subhro Das, Lam M Nguyen, and Tsui-Wei Weng. Label-free concept bottleneck models. *arXiv preprint arXiv:2304.06129*, 2023. 2, 7

[34] Sarah Pratt, Ian Covert, Rosanne Liu, and Ali Farhadi. What does a platypus look like? generating customized prompts for zero-shot image classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15691–15701, 2023. 2

[35] Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K Reddy. Llm-sr: Scientific equation discovery via programming with large language models. *arXiv preprint arXiv:2404.18400*, 2024. 2

[36] Samuel Stevens, Jiaman Wu, Matthew J Thompson, Elizabeth G Campolongo, Chan Hee Song, David Edward Carlyn, Li Dong, Wasila M Dahdul, Charles Stewart, Tanya Berger-Wolf, et al. Bioclip: A vision foundation model for the tree of life. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19412–19424, 2024. 3

[37] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. *ICCV*, 2023. 1, 3, 6

[38] Luming Tang, Davis Wertheimer, and Bharath Hariharan. Revisiting pose-normalization for fine-grained few-shot recognition. In *CVPR*, 2020. 2

[39] United States Census Bureau. American community survey 5-year estimates. https://www.census.gov/programs-surveys/acs, 2024. Accessed: 2024-10-01. 6

[40] John E Vargas-Munoz, Shivangi Srivastava, Devis Tuia, and Alexandre X Falcao. Openstreetmap: Challenges and opportunities in machine learning and remote sensing. *IEEE Geoscience and Remote Sensing Magazine*, 9(1):184–199, 2020. 5, 6

[41] Michael Xie. *MAPPING POVERTY WITH SATELLITE IMAGERY*. PhD thesis, STANFORD UNIVERSITY, 2017. 3

[42] Yue Yang, Artemis Panagopoulou, Shenghao Zhou, Daniel Jin, Chris Callison-Burch, and Mark Yatskar. Language in a bottle: Language model guided concept bottlenecks for interpretable image classification. In *CVPR*, 2023. 2, 7

[43] Christopher Yeh, Chenlin Meng, Sherrie Wang, Anne Driscoll, Erik Rozi, Patrick Liu, Jihyeon Lee, Marshall Burke, David B Lobell, and Stefano Ermon. Sustainbench: Benchmarks for monitoring the sustainable development goals with machine learning. *arXiv preprint arXiv:2111.04724*, 2021. 6

[44] Xixian Yong and Xiao Zhou. MuseCL: Predicting urban socioeconomic indicators via multi-semantic contrastive learning. *CoRR*, 2024. 3

[45] Bolei Zhou, Yiyou Sun, David Bau, and Antonio Torralba. Interpretable basis decomposition for visual explanation. In *ECCV*, 2018. 2