Active Learning for Neurosymbolic Program Synthesis

CELESTE BARNABY, University of Texas at Austin, USA
QIAOCHU CHEN, New York University, USA
RAMYA RAMALINGAM, University of Pennsylvania, USA
OSBERT BASTANI, University of Pennsylvania, USA
IŞIL DILLIG, University of Texas at Austin, USA

The goal of *active learning for program synthesis* is to synthesize the desired program by asking targeted questions that minimize user interaction. While prior work has explored active learning in the purely symbolic setting, such techniques are inadequate for the increasingly popular paradigm of *neurosymbolic program synthesis*, where the synthesized program incorporates neural components. When applied to the neurosymbolic setting, such techniques can —and, in practice, do — return an unintended program due to mispredictions of neural components. This paper proposes a new active learning technique that can handle the unique challenges posed by neural network mispredictions. Our approach is based upon a new evaluation strategy called *constrained conformal evaluation (CCE)*, which accounts for neural mispredictions while taking into account user-provided feedback. Our proposed method iteratively makes CCE more precise until all remaining programs are guaranteed to be observationally equivalent. We have implemented this method in a tool called SmartLabel and experimentally evaluated it on three neurosymbolic domains. Our results demonstrate that SmartLabel identifies the ground truth program for 98% of the benchmarks, requiring under 5 rounds of user interaction on average. In contrast, prior techniques for active learning are only able to converge to the ground truth program for at most 65% of the benchmarks.

CCS Concepts: • **Software and its engineering** → *Automatic programming*.

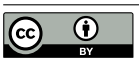
Additional Key Words and Phrases: Program Synthesis, Active Learning, Neurosymbolic Synthesis, Conformal Prediction

ACM Reference Format:
Celeste Barnaby, Qiaochu Chen, Ramya Ramalingam, Osbert Bastani, and Işıl Dillig. 2025. Active Learning for Neurosymbolic Program Synthesis. *Proc. ACM Program. Lang.* 9, OOPSLA2, Article 324 (October 2025), 45 pages. https://doi.org/10.1145/3763102

1 Introduction

Neurosymbolic learning refers to techniques that combine neural networks and symbolic reasoning to learn new concepts in an interpretable and data-efficient manner. A particular instance of this framework is *neurosymbolic program synthesis*, which learns programmatic representations that incorporate neural networks [14]. Typically, these programmatic representations are expressed in a neurosymbolic domain-specific language (DSL) consisting of standard programming constructs (e.g., conditionals, combinators etc.) as well as pre-trained neural networks that convert high-dimensional data (e.g., images) to a more structured symbolic representation. In recent years, neurosymbolic

Authors' Contact Information: Celeste Barnaby, University of Texas at Austin, Austin, USA, celestebarnaby@utexas.edu; Qiaochu Chen, New York University, New York, USA, qc1127@cs.nyu.edu; Ramya Ramalingam, University of Pennsylvania, Philadelphia, USA, ramya23@seas.upenn.edu; Osbert Bastani, University of Pennsylvania, Philadelphia, USA, obastani@seas.upenn.edu; Işıl Dillig, University of Texas at Austin, Austin, USA, isil@cs.utexas.edu.



ACM 2475-1421/2025/10-ART324
https://doi.org/10.1145/3763102

program synthesis has found numerous applications in image processing and search [7, 8, 22], data extraction [15, 17, 78], and question answering [16, 31, 73].

Although neurosymbolic representations are generally more interpretable than purely neural approaches, a fundamental challenge is ensuring that the learned program is the intended one. While the user can inspect the synthesized program to evaluate its correctness, this step can pose a significant hurdle for end-users who do not possess the necessary programming expertise. Prior work on program synthesis [24, 38, 39, 50, 80] has aimed to address this issue through *active learning* techniques that interactively query the user to clarify ambiguities. The key idea behind these techniques is to select questions that will minimize the number of user interactions, while ensuring that the correct program is eventually returned by the synthesizer.

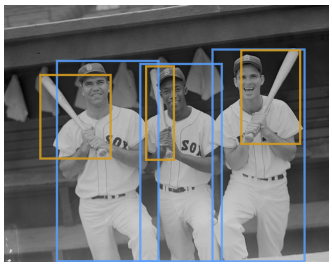
However, when applied to the neurosymbolic setting, these techniques may return an unintended program. To see why, consider the image editing task from prior work [7], where the goal is to learn a neurosymbolic program that generalizes from a small set of input-output examples to edit a large collection of images. For illustration, suppose the user wants to extract individual photos of people holding baseball bats from a dataset of images, each containing multiple individuals. This task may be achieved using the program in Figure 1a, written in a neurosymbolic DSL. Here, the **Is** operator is implemented using a neural object detector. The program identifies all `person` objects in an image that are holding `baseball_bat` objects, and applies a crop operation to those individuals. The notion of "holding" is approximated by the spatial relationship NextTo. To synthesize this program, active learning techniques iteratively query the user for labeled examples (e.g., providing the expected output for a given image) until the system is confident in the inferred logic. Returning to our example, an active learner might ask the user to label the image in Figure 1b by providing the three cropped photos in Figure 1c. However, for this particular image, a state-of-the-art neural classifier [65] fails to detect the baseball bat held by the person in the middle. As a result, when the correct program is executed on the input image, its output does not match the user-provided examples, leading the system to incorrectly reject it as an invalid hypothesis despite it being the desired ground-truth program.

Motivated by this problem, this paper proposes a new active learning framework targeting neurosymbolic programs. Our proposed method deals with possible mispredictions of neural components using *conformal prediction* [2, 5], a principled technique for providing reliable measures of uncertainty for machine learning models. Rather than producing a single prediction, conformal prediction yields a *set* of predictions such that the true label is contained within this prediction set with very high probability. In the context of neurosymbolic programming, we can use these prediction sets to define *conformal semantics* for programs such that every program returns an *output set* that is very likely to contain the ground truth label [60]. A program P is considered to be a possible solution to the learning task if P is consistent with the specification under the conformal semantics. The key advantage of this approach is that it significantly reduces the risk of discarding the correct program despite neural mispredictions.

When using conformal semantics, active learning becomes even more critical because the inherent flexibility of returning a *set* of predictions leads to an explosion in the number of potential solutions. A neurosymbolic synthesis problem can easily admit an intractable number of programs consistent with the conformal semantics, making it impractical for users to manually evaluate all possibilities. The role of active learning is to guide the user through a sequence of targeted questions that gradually refine the hypothesis space of programs by narrowing down the conformal prediction sets. Each round of interaction refines the program's conformal semantics, either by enhancing the task specification or by confirming the ground-truth label for a neural prediction. Thus, as user interaction progresses, program evaluation needs to account for both the results of conformal

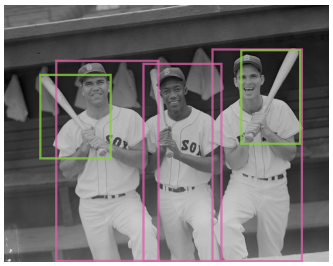
$$\{\mathsf{Find}(\mathbf{Is}(\mathsf{Object}(\mathtt{baseball\_bat})), \mathbf{Is}(\mathsf{Object}(\mathtt{person})), \mathsf{NextTo}) \rightarrow \mathsf{Crop}\}$$

(a) A program in a neurosymbolic image editing DSL.

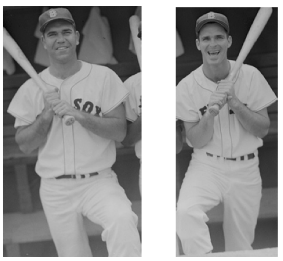


(b) Ground truth object detections.

(c) Output provided by the user.



(d) Objects detected by the neural network.



(e) Output of the above program.

Fig. 1. (a) A program for cropping people who hold baseball bats. (b) The ground truth object detections, with people in blue and baseball bats in orange. (c) An example output provided by the user. (d) The objects detected by the neural network. (e) The output of the program in (a) when executed on the image.

prediction as well as any user feeback provided thus far. We refer to this type of program evaluation strategy as *constrained conformal evaluation (CCE)*.

As illustrated in Figure 2, each round of user interaction chooses a question to ask to the user, with the goal of shrinking the hypothesis space as much as possible. In each round of user interaction, programs are eliminated from the hypothesis space either because the user provides a new input-output example or because the CCE results have become more precise. Upon refinement of the hypothesis space, a *distinguishability* check is performed to identify pairs of programs that *could* have different CCE results if we continued to query the user. If such programs exist, user interaction continues and active learning identifies a new question. Active learning terminates when all remaining programs in the hypothesis space are observationally equivalent.

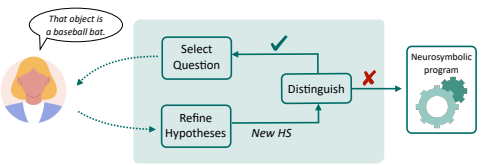


Fig. 2. Overview of our approach.

A key challenge in realizing the active learning framework from Figure 2 is that its key components (namely, Select Question, Refine Hypotheses, and Distinguish) require performing CCE many times, which can be very expensive. In particular, under conformal semantics, expressions evaluate to *sets of values*, so CCE can require an exponential number of computations with respect to the cardinality of these sets. Our method addresses this problem using two key ideas. First, it uses bidirectional abstract interpretation to perform CCE in a practical way. At a high level, the idea is to use the output examples provided by the user to infer abstract values for program sub-expressions. These abstract values can then be used to filter infeasible conformal prediction results, leading to smaller sets. Second, when selecting what questions to ask the user, our method uses a bounded

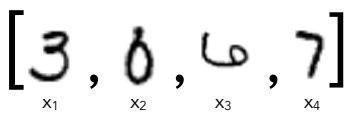


Fig. 3. List of hand-drawn digits.

$$P_1 := \lambda\, l.\ \texttt{fold inc 0 (filter } (\lambda x.\ x > 5)\texttt{(map } \textbf{\texttt{toDigit}}\ l))$$
$$P_2 := \lambda\, l.\ \texttt{fold inc 0 (filter } (\lambda x.\ 6 < x < 9)\texttt{(map } \textbf{\texttt{toDigit}}\ l))$$
$$P_3 := \lambda\, l.\ \texttt{fold inc 0 (filter } (\lambda x.\ x < 4)\texttt{(map } \textbf{\texttt{toDigit}}\ l))$$
$$P_4 := \lambda\, l.\ \texttt{fold inc 0 (filter } (\lambda x.\ 2 < x < 8)\texttt{(map } \textbf{\texttt{toDigit}}\ l))$$

Fig. 4. Hypothesis space $\mathcal{P}$.

form of conformal evaluation to over-approximate the pruning power of each question, resulting in a much more practical question selection algorithm.

We have implemented the proposed approach in a tool called SmartLabel and evaluated it on 112 benchmarks spanning three neurosymbolic domains: (1) batch image editing, (2) visual arithmetic, and (3) image search for visual concept learning. A key highlight of our evaluation is that SmartLabel can successfully identify the desired program for 98% of the benchmarks, whereas prior active learning techniques that do not use conformal semantics fail to produce the desired program for around 35% of the benchmarks. Another key highlight of our evaluation is that SmartLabel converges to the desired program in under 5 rounds of interaction on average. Additionally, several ablation studies confirm the effectiveness of our proposed algorithmic optimizations.

To summarize, this paper makes the following key contributions:

- We define the *neurosymbolic active learning* problem and propose the first algorithm for solving it.
- We introduce *constrained conformal evaluation (CCE)* as a new type of program semantics that takes into account both user feedback and prediction sets obtained through conformal prediction. Furthermore, we present a practical CCE technique that uses bidirectional abstract interpretation.
- We implement our approach in a new tool called SmartLabel and instantiate it on three neurosymbolic domains. Our experiments on 112 tasks show that SmartLabel can identify the desired program for 98% of the benchmarks, requiring an average of 4.9 rounds of user interaction.

2 Overview

In this section, we provide a high-level overview of our approach using a simple example. Specifically, suppose that a user is examining a hand-written ledger and wants to count the number of transactions greater than 5 dollars. This task may be automated using the program P_1 in Figure 4, which uses standard combinators such as `map`, `fold`, and `filter`, along with a neural perception component, **`toDigit`**, which converts an image to a digit. Consider the input list I in Figure 3. Note that the true label for x_2 in I is ambiguous; if the user believes the label is 0, then the desired output is 2. However, if **`toDigit`** classifies x_2 as 8, then $P_1(I)$ will return 3, disqualifying it as a solution. In this case, the synthesizer might fail to return any program, or it may produce an erroneous program that doesn't align with the user's expectations.

Solution: conformal semantics. As discussed in Section 1, a viable solution is to use *conformal prediction (CP)* [2] to ensure the ground truth label is included in the result with high probability. CP generates prediction sets for unlabeled data by calculating nonconformity scores based on previously labeled data, determining which labels to include to maintain a specified confidence level. In our example, given the input list from Figure 3, CP would produce the prediction sets [{3}, {0, 8}, {6, 9}, {7}], indicating the model's uncertainty about the second and third digits. This uncertainty impacts the output of P_1 from Figure 4: if the second digit is 0, P_1 returns 2; if it is 8, it returns 3. Thus, under *conformal semantics*, the output of P_1 is the set $\{2, 3\}$. Since this set includes the user-provided label 2, P_1 remains consistent with the input-output example and is not pruned.

Need for active learning. While the intended program is no longer pruned from the solution space, a new problem arises: *many* other programs are also consistent with the input-output examples under conformal semantics. In our running example, any other program whose output set

contains 2 (e.g., P_2, P_3, P_4 in Figure 4) is also a potential solution. This issue already exists in standard Programming-By-Example (PBE) scenarios, where input-output examples do not fully capture the program's intended behavior. To deal with this issue, prior work has proposed *active learning* techniques that help users resolve such ambiguity through user interaction [50, 80]. However, these techniques only deal with ambiguity in the specification. In the neurosymbolic setting, another significant source of ambiguity involves the ground truth labels of neural components. Existing techniques for active learning fail to handle this second class of ambiguities and may prune the intended program. In the remainder of this section, we explain how our proposed method interacts with the user to correctly resolve both types of ambiguities.

Checking ambiguities. We first need a method to determine whether there are any remaining ambiguities in the solution space. Intuitively, the solution space $\mathcal{P}$ is *ambiguous* if it contains a pair of *distinguishable* programs (P, P'), meaning they could produce different outputs under the *ground truth semantics*. For example, in the hypothesis space $\mathcal{P}$ from Figure 4, the pair (P_1, P_2) is distinguishable because P_1's output set on I is $\{2, 3\}$, while P_2's output set is $\{1, 2\}$. Thus, it is possible that $P_1(I) = 2$ and $P_2(I) = 1$, meaning they differ under the ground truth. P_1 and P_4 are also distinguishable even though both produce $\{2, 3\}$ under conformal semantics, as P_1 could produce 2 under the ground truth semantics, while P_4 could produce 3.

Question selection. Since the solution space in our example is ambiguous, we query the user. User queries can either request a new input-output example for the target function or ask the user to label an image. Our approach selects a question q based on its *pruning power*—the fraction of programs that will be pruned from the hypothesis space for the worst possible answer to q. For instance, consider a question q_1 asking the user to label x_2 from Figure 3. If the user labels x_2 as 0, P_2 will be pruned from the program space, while if the user labels x_2 as 8, P_1 and P_3 will be pruned. Hence, q_1 has a pruning power of $\min(0.25, 0.5) = 0.25$. In contrast, consider question q_2 asking the user to label x_3. If the answer is 9, then all programs can still output 2, meaning that q_2 has a pruning power of 0. Thus, q_1 is the preferable question.

Hypothesis space refinement. After the user responds to q_1 with 0, we need to determine which programs in the hypothesis space are still viable. To do this, we evaluate each program $P \in \mathcal{P}$ under the conformal semantics while incorporating all user feedback—a process called *constrained conformal evaluation (CCE)*. For instance, before the user answers q_1, the result of CCE on P_2 for I is $\{1, 2\}$, as x_2 could be between 6 and 9. After the user labels x_2 as 0, the result becomes $\{1\}$. Since the output no longer includes 2, P_2 is inconsistent with the input-output example and is pruned from the hypothesis space. At this point, however, there is still ambiguity because P_1, P_3, and P_4 can all output 2 but they are not equivalent. Hence, our method repeats this process of checking for ambiguities and generating new questions until all remaining programs are equivalent.

Challenges. As discussed above, our procedure involves three main components: checking for ambiguities, question selection, and hypothesis space refinement. Among these components, checking for ambiguities often takes negligible time because we just need to find a single pair of programs that are distinguishable (see Section 6.6). Meanwhile, hypothesis space refinement and question selection take considerable time due to the need to perform CCE on all pairs of programs and inputs. Unfortunately, performing CCE is computationally quite expensive – for instance, computing the conformal output of $P_1(I)$ requires evaluating P_1 on four possible combinations: $\{3\} \times \{0, 8\} \times \{6, 9\} \times \{7\}$. While this is manageable for small examples, it becomes impractical for larger inputs or prediction sets. For example, with list of 5 images each having a prediction set of size 4, conformal evaluation would need to consider 1024 possibilities. Our algorithm deals with this challenge through two key algorithmic optimizations, one primarily targeting hypothesis

space refinement and the other targeting question selection. Next, we give an overview of these two key optimizations.

Practical CCE for hypothesis space refinement. To make hypothesis space refinement practical, we leverage a key observation: *the primary use of CCE is to check whether or not a program is consistent with the input-output examples.* Our method uses this observation to make CCE more scalable. By reasoning backwards from the desired output, we can infer constraints on subexpressions and thereby to reduce the size of the prediction sets during conformal evaluation.

As an example, consider checking whether P_3 is consistent with the user's IO example. To perform CCE more efficiently, our method first performs abstract interpretation in the forward direction to compute abstract values for each subexpression, as shown in the annotated abstract syntax tree (AST) in Figure 5. Here, each node is annotated by its abstract value $[\alpha_1, \ldots, \alpha_n]$, where each α_i is either an interval or an *optional* interval $[l, u]?$, indicating the value may or may not exist at that position in the list. The leaf node annotations reflect the conformal prediction results (e.g., the set $\{0, 8\}$ is abstracted as $[0, 8]$), and abstract values of internal nodes are obtained by applying abstract transformers (e.g., resulting in the abstract value $[1, 2]$ for the root).

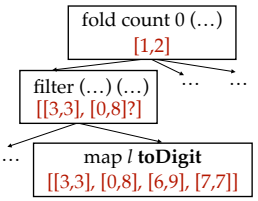


Fig. 5. The annotated AST of P_3 after forward abstract interpretation.

Next, our method performs backward reasoning from the output to tighten these annotations. For instance, in order for P_3 to output 2, there must be exactly 2 items less than 4 in the input. Since x_3 and x_4 are certainly ≥ 4, we can infer x_2 must not have been filtered; hence $x_2 < 4$. As a result, we obtain the strengthened annotations shown in Figure 6. Intuitively, these annotations correspond to necessary conditions that each element must satisfy in order for the program to be consistent with the input-output examples. Thus, when performing CCE, we can filter evaluation results that do not satisfy the inferred necessary condition. For example, when evaluating P_3 on I, we need only consider the 2 possible ground truth values in $\{3\} \times \{0\} \times \{6, 9\} \times \{7\}$. In practice, this approach significantly reduces the sizes of output sets when performing CCE.

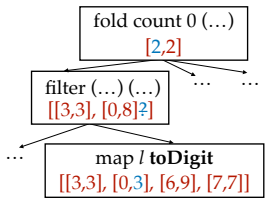


Fig. 6. The annotated AST of P_3 after backward abstract interpretation.

Minimizing CCE for question selection. Recall that, during question selection, we need to compute the pruning power of each question, which naïvely requires performing CCE on all (program, input) pairs. To make question selection more practical, we leverage the following key observations: *(1) Any evaluation strategy that under-approximates the CCE result can be used to derive an upper bound on the pruning power of a question. (2) If the upper bound for a question q is lower than that of the true pruning power of a previously encountered question, we can discard it and avoid performing CCE for q.* Based on these observations, our method first uses a more lightweight method, called *bounded conformal evaluation (BCE)*, to derive an upper-bound on the pruning power of every question and uses them to reduce the set of questions for which CCE must be performed.

For instance, consider again question q_2 asking the user to label the ground truth value of x_3. To compute an upper bound on q_2's pruning power in a lightweight manner, we perform a bounded form of CCE where we restrict intermediate results to sets of size at most k through sampling. For instance, setting $k = 1$ in this example and sampling the value 0 from the prediction set of x_2, we determine that the only program that could be pruned by q_2 is P_4, which gives an upper bound of 0.25 on the pruning power of q_2. Now, if we have previously computed the true pruning power of

q_1 as 0.25, we need not consider q_2, since it cannot possibly be a strictly better question than q_1. As we show experimentally, this strategy greatly reduces user interaction time.

3 Problem Formulation

3.1 Neurosymbolic Programs

We consider a family of *neurosymbolic* programming languages defined by the *meta-grammar* in Figure 7, which is parametrized over symbolic functions $\mathcal{S}$ and neural networks $\mathcal{N}$. Every program defines a function called f_{synth} in a functional language with expressions, let statements, and sequencing. Expressions include conditionals of the form $E\ ?\ E : E$ as well as compositions of symbolic and neural components. In particular, $f_s(E_1, \ldots, E_n)$ denotes the application of a symbolic function $f_s \in \mathcal{S}$ to expressions $E_1, \ldots, E_n$. On the other hand, $f_n(x)$ performs forward propagation on input x using a neural network $f_n \in \mathcal{N}$.

$$
\begin{array}{lcl}
P & := & \text{def } f_{\text{synth}}(x) = S \\
S & := & E \mid \texttt{let } x = E \mid S; S \\
E & := & c \mid x \mid f_s(E, \ldots, E) \mid f_n(x) \mid E\ ?\ E : E
\end{array}
$$

$f_s \in$ Symbolic functions $\quad f_n \in$ Neural networks

Fig. 7. Meta-grammar for DSL.

While the distinction between neural and symbolic components may seem minor in terms of syntax, it creates a significant difference in semantics. Symbolic functions have precisely defined semantics and always produce *accurate* results. In contrast, neural components f_n *approximate* an oracle function $\hat{f}_n$ and may produce inaccurate results for some inputs (i.e., there may exist an input x such that $f_n(x) \neq \hat{f}_n(x)$). To capture this difference, we define two types of semantics for neurosymbolic DSLs: *evaluation* semantics $[\![P]\!]$ and *ground truth* semantics $(\!|P|\!)$. These differ only in their treatment of neural components: $[\![f_n]\!](x)$ represents the output of a forward pass through the neural network f_n, while $(\!|f_n|\!)(x)$ consults an external oracle $\hat{f}_n$ to return the *true label* for x.

3.2 Neurosymbolic Learning and Conformal Semantics

This paper is concerned with *neurosymbolic learning* from examples. That is, given a set $Z = \{(I, O)\} \subseteq \mathcal{I} \times \mathcal{O}$ (where $\mathcal{I}$ is an input space and $\mathcal{O}$ is an output space), we wish to learn a program P in a neurosymbolic DSL such that P is consistent with Z under the *ground truth semantics*, i.e.:

$$\forall (I, O) \in Z.\ (\!|P|\!)(I) = O \tag{1}$$

However, since the ground truth semantics requires consulting an external oracle (e.g., a human), we cannot easily check whether a program P constitutes a valid solution to the learning problem. As a consequence, running a traditional synthesizer with the ground truth semantics would require the user to provide an enormous number of ground truth labels.

A natural strategy for minimizing this workload is to leverage active learning. To do so, we need to identify programs that are consistent with the input examples Z and ground truth labels for perception. In this section, we focus on the former and discuss the latter later. A naïve approach is to approximate the ground truth semantics via the evaluation semantics, and apply an existing active learning strategy for PBE. That is, we could formulate the learning problem as that of finding a program P that is consistent with Z under the evaluation semantics, such that

$$\forall (I, O) \in Z.\ [\![P]\!](I) = O \tag{2}$$

However, as discussed in previous sections, this alternative formulation has a significant drawback: because the evaluation semantics may be incorrect, a solution to Equation 1 may *not* be a solution to Equation 2. This means that the alternative problem formulation would inherently allow incorrect solutions, while ruling out the desired solution.

Recent work has proposed to use *conformal prediction* to address this problem [60]. At a high level, conformal prediction [2, 67] replaces each neural component $f_n : \mathcal{X} \to \mathcal{Y}$ with a *conformal*

predictor $\tilde{f}_n : \mathcal{X} \to 2^{\mathcal{Y}}$ that predicts sets of labels $\tilde{O} \subseteq \mathcal{Y}$. Intuitively, $\tilde{f}_n(x)$ represents the set of all plausible labels for input x. More precisely, conformal prediction aims to ensure that the ground truth label is contained in the prediction set. This property is known as *coverage*, and prior work in the conformal prediction literature [60] can provide coverage with bounded probability under the standard i.i.d. assumption. In the remainder of this paper, we assume access to a conformal predictor that can guarantee coverage, and we utilize such a predictor to define the *conformal semantics* of any neurosymbolic DSL (formalized in the next subsection). In more detail, given an input I, the conformal semantics $[\![P]\!](I)$ yields a *set* $\tilde{O}$ of outputs such that $(\!|P|\!)(I) \in [\![P]\!](I)$. In other words, the conformal semantics generates a set of outputs that includes the ground truth, and we define our learning problem in terms of these conformal semantics. Section 5 provides details about the conformal prediction technique used in our implementation.

Definition 3.1. **(Neurosymbolic learning from examples (NSLE))** Given a set of input-output examples Z, the goal of neurosymbolic learning is to find a set of programs $\mathcal{P}$ satisfying:

$$P \in \mathcal{P} \iff \forall (I, O) \in Z.\, O \in [\![P]\!](I) \tag{3}$$

In other words, the goal of NSLE is to identify all programs consistent with the examples under the *conformal* semantics. Each such program is referred to as a *solution* to the NSLE problem. While our formulation of NSLE is guaranteed to include the desired solution, it may also admit *many* others. As a result, users must manually identify the correct program from a potentially large set of candidates—a process that can be burdensome, especially without expertise in the underlying DSL. To mitigate this, we define the *active learning for NSLE* problem, which aims to efficiently guide the user toward the intended solution.

3.3 Constrained Conformal Semantics

At a high level, the goal of active learning is to refine the conformal semantics until the ground truth program is found. Intuitively, the approach starts with the vanilla conformal semantics, but each round of user interaction makes the conformal semantics more precise, making the output sets smaller while still containing the ground truth. We refer to the refined semantics that incorporates user feedback as the *constrained conformal semantics*, and we formally define *user feedback* to precisely characterize this alternative semantics:

Definition 3.2. **(User feedback)** Let $\mathcal{I}$ be a finite input space[1] on which programs will be executed. User feedback $\mathcal{F}$ is a set of *labels* (f, I, O) where $f \in \mathcal{N} \cup \{f_{\text{synth}}\}$, $I \in \mathcal{I}$, and $O = (\!|f|\!)(I)$.

In this definition, a label can take two forms: If $f = f_{\text{synth}}$, the label corresponds to an input-output example for the function to be synthesized. On the other hand, for $f \in \mathcal{N}$, the label specifies the ground truth semantics of neural component f for a specific input. Given user feedback $\mathcal{F}$, we use the notation $\Phi_{\mathcal{F}}$ to denote the formula representation of $\mathcal{F}$, i.e.,

$$\Phi_{\mathcal{F}} = \bigwedge_{(f,I,O) \in \mathcal{F}} f(I) = O$$

For notational convenience, we often leave $\mathcal{F}$ implicit and denote user feedback simply by Φ. We assume that the user always gives feedback consistent with the ground truth.

Next, we define *constrained conformal semantics* to evaluate programs while incorporating user feedback. Given a program P and input I, constrained conformal evaluation uses the user-provided label for evaluating an expression E when available, and performs standard conformal evaluation otherwise. Formally, Figure 8 presents the constrained conformal semantics using judgments of the

[1] As standard in prior work, we assume a finite set of inputs because, in the PBE setting, the synthesized program is used to automate a task on a large but fixed set of inputs.

$$\textsc{Neural}\ \frac{\sigma(x) = I \quad \Delta = \{(O, f_n(I) = O) \mid O \in \tilde{f_n}(I) \wedge \mathsf{SAT}(f_n(I) = O \wedge \Phi)\}}{\sigma \vdash f_n(x) \Downarrow_\Phi \Delta}$$

$$\textsc{Symb}\ \frac{\begin{array}{c}\sigma \vdash E_1 \Downarrow_\Phi \Delta_1 \quad \ldots \quad \sigma \vdash E_n \Downarrow_\Phi \Delta_n, \\ \Delta = \{([\![f_s]\!](O_1, \ldots, O_n), \bigwedge_i \varphi_i) \mid ((O_1, \varphi_1), \ldots (O_n, \varphi_n)) \in \Delta_1 \times \ldots \times \Delta_n \wedge \mathsf{SAT}(\Phi \wedge \bigwedge_i \varphi_i)\}\end{array}}{\sigma \vdash f_s(E_1, \ldots, E_n) \Downarrow_\Phi \Delta}$$

$$\textsc{Assign}\ \frac{\sigma \vdash E \Downarrow_\Phi \Delta}{\sigma \vdash \texttt{let}\ x = E; S \Downarrow_\Phi \Delta'} \qquad \textsc{Lookup}\ \frac{}{\sigma \vdash x \Downarrow_\Phi \sigma(x)} \qquad \textsc{Prog}\ \frac{\sigma(x) = I \quad \sigma \vdash S \Downarrow_\Phi \Delta \quad \tilde{O} = \{O \mid (O, \varphi) \in \Delta\}}{\sigma \vdash \mathsf{def}\ f_{\mathsf{synth}}(x) = S \Downarrow_\Phi \tilde{O}}$$

Fig. 8. Constrained conformal semantics. Conditionals are omitted because they are a special case of Symb.

form $\sigma \vdash S \Downarrow_\Phi \tilde{O}$, indicating that S evaluates to the output set $\tilde{O}$ under the valuation σ and user feedback Φ (represented as a formula).

The first rule, Neural, shows how to evaluate a neural expression $f_n(x)$ under constrained conformal semantics. It first looks up the value I for x under σ and performs conformal prediction to obtain the set of possible outputs $\tilde{O}$. The result is then filtered to retain only outputs consistent with the user feedback, i.e., those for which $f_n(I) = O \wedge \Phi$ is logically satisfiable.[2] As discussed in Section 3.2, conformal prediction guarantees that $\tilde{O}$ will contain the user feedback with bounded, user-controllable probability.

The second rule, Symb, handles symbolic expressions by evaluating each nested expression E_i under constrained conformal semantics and applying f_s to each combination of results, filtering out combinations with unsatisfiable constraints. The final rule, Prog, shows how to evaluate the entire program under user feedback. This rule evaluates the function body using the other rules to obtain the result Δ, and then discards the constraints to produce the final output set.

Definition 3.3. **(Constrained conformal evaluation)** Let P be a program, I an input, and let Φ denote user feedback. We say that P conformally evaluates to $\tilde{O}$ on I under user feedback Φ, denoted $[\![P]\!]_\Phi(I) = \tilde{O}$, if and only if $[x \mapsto I] \vdash P \Downarrow_\Phi \tilde{O}$.

Note that a program P may produce an output set inconsistent with a user-provided label. Therefore, we define a notion of consistency with user feedback:

Definition 3.4. **(Consistency with user feedback)** We say that a program P is *consistent* with user feedback $\Phi_\mathcal{F}$, denoted $P \models \Phi_\mathcal{F}$, iff $\forall\ (f_{\mathsf{synth}}, I, O) \in \mathcal{F}.\ \ O \in [\![P]\!]_{\Phi_\mathcal{F}}(I)$.

In other words, a program P is consistent with user feedback if, for all input-output examples (I, O) provided by the user, P can return O on I under the conformal semantics.

Example 3.5. Let $l = [x_1, x_2, x_3]$ be an input list of hand-written digits for which the ground truth label is $[2, 4, 9]$ and whose conformal prediction result is $[\{2, 7\}, \{4\}, \{8, 9\}]$. Consider the following pairs of user feedback and programs:

$$\Phi_1 = (f_{\mathsf{synth}}(I) = 15) \wedge (\texttt{toDigit}(x_1) = 2) \qquad P_1 = \lambda x.\texttt{fold}\ (\texttt{map}\ x\ \texttt{toDigit})\ 0\ \texttt{sum}$$
$$\Phi_2 = (f_{\mathsf{synth}}(I) = 15) \wedge (\texttt{toDigit}(x_3) = 9) \qquad P_2 = \lambda x.\texttt{fold}\ (\texttt{map}\ x\ \texttt{toDigit})\ 1\ \texttt{sum}$$

[2]While we formalize this consistency check as the satisfiability of a logical formula, this check does not require an SMT solver. In particular, these formulas can be viewed as a set of input-output examples for $f_n \in \mathcal{N}$. Thus, this consistency check may be implemented by querying whether an input I maps to output O in a hash map designated for f_n.

Here, for P_1, we have $P_1 \models \Phi_1$ and $P_1 \models \Phi_2$. However, for P_2, we have $P_2 \models \Phi_1$ but $P_2 \not\models \Phi_2$. This is because P_1 and P_2 evaluate to the following values under the constrained conformal semantics:

$$[\![P_1]\!]_{\Phi_1}(I) = \{14, 15\} \quad [\![P_1]\!]_{\Phi_2}(I) = \{15, 20\} \quad [\![P_2]\!]_{\Phi_1}(I) = \{15, 16\} \quad [\![P_2]\!]_{\Phi_2}(I) = \{16, 21\}$$

3.4 Active Learning Problem

In this section, we formally define the *active learning* problem addressed in this paper.

Definition 3.6. **(Hypothesis space)** Let $\mathcal{P}$ be the set of all possible programs, and let Φ be the current user feedback. The hypothesis space defined by Φ, denoted $\mathcal{P} \downarrow \Phi$, is the set $\{P \in \mathcal{P} \mid P \models \Phi\}$.

Definition 3.7. **(Question)** A question q is a tuple (f, I) where $f \in \mathcal{N} \cup \{f_{\mathsf{synth}}\}$ and $I \in \mathcal{I}$. We say that an answer a to a question (f, I) is valid iff $a = (\!|f|\!)(I)$.

Note that there are two types of questions: those involving f_{synth} ask the user to provide a new input-output specification of the function being synthesized. In contrast, those involving neural components $f \in \mathcal{N}$ ask the user to provide the ground truth label for some perception task. Given a question $q = (f, I)$ with a *valid* answer a, we use $\mathsf{Feedback}(q)$ to denote the feedback $f(I) = a$, and $\mathsf{Possible}(q)$ to represent the set of possible user feedback formulas for q. This terminology naturally extends to a question space $\mathcal{Q}$.

Definition 3.8. **(Indistinguishable Programs)** Two neurosymbolic programs P_1 and P_2 are *indistinguishable* under feedback Φ, denoted $P_1 \simeq_\Phi P_2$, if for all inputs I in the input space and for any set of questions Q not already answered by Φ, we have:

$$\forall \Phi' \in \mathsf{Possible}(Q).\ [\![P_1]\!]_{\Phi \wedge \Phi'}(I) = [\![P_2]\!]_{\Phi \wedge \Phi'}(I) \tag{4}$$

In other words, programs P_1, P_2 are indistinguishable under Φ if we cannot distinguish their behavior by asking the user more questions. Note that Definition 3.8 *implies* $[\![P_1]\!]_\Phi(I) = [\![P_2]\!]_\Phi(I)$; however, $[\![P_1]\!]_\Phi(I) = [\![P_2]\!]_\Phi(I)$ does *not* imply that the programs are indistinguishable. In particular, two programs that evaluate to the same output set under Φ *could* evaluate to different sets as we strengthen Φ by getting additional feedback from the user.

Example 3.9. Consider two programs $P_1 = \lambda x.f_n^1(x)$, $P_2 = \lambda x.f_n^2(x) + 1$, input space $\{I\}$, and user feedback $\varnothing$. Suppose further that $\tilde{f_n^1}(I) = \{1, 2\}$ and $\tilde{f_n^2}(I) = \{0, 1\}$. Here, we have $[\![P_1]\!]_\Phi(I) = [\![P_2]\!]_\Phi(I) = \{1, 2\}$; but these programs are *distinguishable*. In particular, consider the question (f_n^1, I) and possible answer 2. Then, we have $[\![P_1]\!]_{f_n^1(x)=2}(I) = \{2\}$, but $[\![P_2]\!]_{f_n^1(x)=2}(I) = \{1, 2\}$, so they violate Equation 4. Our definition of indistinguishability requires the programs to remain observationally equivalent under any answers to future questions.

Lemma 3.1. Let P_1, P_2 be a pair of programs such that $P_1 \simeq_\Phi P_2$. Then, assuming Φ represents accurate user feedback, we have $\forall I \in \mathcal{I}.\ (\!|P_1|\!)(I) = (\!|P_2|\!)(I)$ where $\mathcal{I}$ denotes the input space.[3]

This lemma is important because it states that active learning can terminate when all programs in the hypothesis space are indistinguishable from one another.

Definition 3.10. **(Active learning for NSLE).** Let $\mathcal{Q}$ be the space of all possible queries and let P^* be a ground truth program. The goal of the active learning problem is to ask the user a series of questions $q_0, \ldots, q_n$ in $\mathcal{Q}$ such that (1) for $\Phi = \bigwedge_{i=0}^{n} \mathsf{Feedback}(q_i)$, we have $\forall P \in (\mathcal{P} \downarrow \Phi).\ P \simeq_\Phi P^*$; (2) the number of questions asked to the user is minimized.

[3]Proofs of all lemmas and theorems are provided in Appendix A.

```
1:  procedure ActiveLearning(P, Q, I, Z)
    input: Hypothesis space P, question space Q, input
    space Q and IO examples Z.
    output: A program P ∈ P
2:      Φ ← {f_synth(I) = O | (I, O) ∈ Z}
3:      while true do
4:          P ← RefineHS(P, Φ, Z)
5:          P' ← Sample(P)
6:          if ¬Distinguish(P', P \ P', Φ, I, Q) then
7:              break;
8:          q ← SelectQuestion(P, Q)
9:          O ← QueryUser(q)
10:         Φ ← Φ ∧ f(I) = O;  Q ← Q \ (f, I)
11:         if f = f_synth then Z ← Z ∪ {(I, O)}
12:     return P ∈ P
```

Fig. 9. Active learning algorithm.

```
1:  procedure RefineHS(P, Φ, Z)
    input: P is a hypothesis space, Φ is a user
    feedback formula, and Z is the IO examples.
    output: A new hypothesis space P'
2:      P' ← P
3:      for all P ∈ P do
4:          for all (I, O) ∈ Z do
5:              Õ ← CCE(P, Φ, I)
6:              if O ∉ Õ then
7:                  P' ← P' \ P
8:                  break;
9:      return P'
```

Fig. 10. Hypothesis space refinement algorithm.

The first criterion states that when active learning terminates, all remaining programs are observationally equivalent on inputs $\mathcal{I}$ under the ground truth semantics. The second condition focuses on optimality, aiming to minimize the number of questions the user must answer. However, achieving a globally optimal solution is NP-hard [39] without knowing the answer in advance. Like previous work on active learning [39], we focus on *worst-case* rounds of interaction. Our revised problem aims to construct a sequence of questions where each is *locally* optimal, formalized in terms of pruning power:

Definition 3.11. **(Pruning Power)**. Let $\mathcal{P}$ be the hypothesis space, and let $q = (f, I)$ be a question with possible answers $a_1, \ldots, a_n$. Then, the *pruning power* of q modulo hypothesis space $\mathcal{P}$ is:

$$\Pi(q, \mathcal{P}) = \frac{|\mathcal{P}| - \max_i |(\mathcal{P}\downarrow(f(I) = a_i))|}{|\mathcal{P}|}.$$

In other words, the pruning power of a question q is the fraction of programs pruned from the hypothesis space by the *worst* answer to q.

Definition 3.12. **(Revised problem statement)** Let $\mathcal{Q}$ denote all questions and let P^* be a ground truth program. Active learning aims to ask the user a series of questions $q_0, \ldots, q_n$ in $\mathcal{Q}$ such that:

(1) For $\Phi = \bigwedge_{i=0}^{n} \text{Feedback}(q_i)$, we have: $\forall P \in (\mathcal{P} \downarrow \Phi).\ P \simeq_\Phi P^*$
(2) For $\Phi^i = \bigwedge_{j=0}^{i-1} \text{Feedback}(q_j)$ and $\mathcal{Q}^i = \mathcal{Q} \setminus \cup_{j=0}^{i-1} \{q_j\}$, we have $q_i = \underset{q \in \mathcal{Q}^i}{\text{argmax}}\ \Pi(q, \mathcal{P} \downarrow \Phi^i)$

As before, the first criterion states that all remaining programs in the hypothesis space are indistinguishable. The second condition states a *local optimality* requirement, namely that the next question asked to the user should have maximal pruning power among all remaining questions.

Remark. A reasonable alternative to considering the worst-case answer is to reason about the *expected* answer by leveraging prediction probabilities of neural components. While we also considered this alternative, reasoning about expected rounds of interaction does not work well empirically, as we evaluate in Section 6.4. Intuitively, neural networks can sometimes be confidently wrong about their predictions; hence, reasoning about worst-case behavior turns out to be a better objective in practice.

1: **procedure** Distinguish($P', \mathcal{P}, \Phi, \mathcal{I}, \mathcal{Q}$)
input: Program P', hypothesis space $\mathcal{P}$, user feedback Φ, input space $\mathcal{I}$, question space $\mathcal{Q}$.
output: A boolean indicating whether there is any program in $\mathcal{P}$ distinguishable from P'.
2: **for all** $I, P \in \mathcal{I} \times \mathcal{P}$ **do**
3: **if** $\underline{\text{CCE}}(P, I, \Phi) \neq \underline{\text{CCE}}(P', I, \Phi)$ **then return** true
4: **for all** $(q, a) \in \mathcal{Q} \times \text{Answers}(q)$ **do**
5: **if** Distinguish($P', \mathcal{P}, \Phi \wedge (q = a), \mathcal{I}, \mathcal{Q} \setminus q$) **then return** true
6: **return** false

Fig. 11. Procedure for checking distinguishability.

4 Interactive Neurosymbolic Synthesis Algorithm

4.1 Top-Level Algorithm

Our top-level active learning algorithm is presented in Figure 9. This procedure takes four inputs, namely (1) $\mathcal{P}$, which is the hypothesis space and is assumed to contain the ground truth program P^*, (2) the space $\mathcal{Q}$ of all possible questions, (3) the input space $\mathcal{I}$, and (4) a set of initial input-output examples provided by the user. The return value of ActiveLearning is a program that is observationally equivalent to P^* on the input space under the ground truth semantics.

The ActiveLearning procedure gradually grows the user feedback Φ and refines the hypothesis space $\mathcal{P}$ until all pairs of programs in $\mathcal{P}$ are indistinguishable according to Definition 3.8. Initially, the user feedback Φ contains the input-output examples provided by the user (line 2). In each iteration, the algorithm first computes the new hypothesis space by calling RefineHS (shown in Figure 10), which computes $\mathcal{P} \downarrow \Phi$ from Definition 3.6 with some optimizations discussed in the next subsection. Next, at line 5, it samples a program P' from the hypothesis space and then calls the Distinguish procedure (presented in Figure 11) to check whether there exists any program in $\mathcal{P}$ that is distinguishable from P' according to Definition 3.6. If no distinguishable programs exist, the procedure terminates. Otherwise, it selects a question using SelectQuestion (explained in Section 4.3) to maximize pruning power, and the user's answer is added to the feedback.

The top-level algorithm relies on three auxiliary procedures: RefineHS, Distinguish, and SelectQuestion. Since the first two are straightforward, we describe them here, deferring the discussion of SelectQuestion to Section 4.3. As shown in Figure 10, RefineHS iterates over all programs in the hypothesis space, checking if each $P \in \mathcal{P}$ satisfies the feedback Φ using the CCE procedure for constrained conformal evaluation. If any input-output example $(I, O) \in Z$ is not included in the CCE result, P is pruned from the hypothesis space. The Distinguish procedure is presented in Figure 11 and checks if there is a program $P \in \mathcal{P}$ that is distinguishable from P'. It first examines whether any input $I \in \mathcal{I}$ causes CCE to produce different results for P and P'. If so, the algorithm returns true. Otherwise, it considers all possible question-answer pairs (q, a) and checks if any program $P \in \mathcal{P}$ could be distinguished from P' based on the user's response.

Lemma 4.1. Let $\mathcal{P}$ be a hypothesis space and let P' be a program randomly sampled from $\mathcal{P}$. For any user feedback Φ, input space $\mathcal{I}$, and question space $\mathcal{Q}$, we have $\forall P_1, P_2 \in \mathcal{P}.\ P_1 \simeq_\Phi P_2$ if and only if Distinguish($P', \mathcal{P} \setminus P', \Phi, \mathcal{I}, \mathcal{Q}$) returns false.

Theorem 4.1. Let P^* be the ground truth program, and let P be the program returned by the ActiveLearning procedure. Then, assuming $P^* \in \mathcal{P}$, we have $\forall I \in \mathcal{I}.\ (\!|P|\!)(I) = (\!|P^*|\!)(I)$.

Remark. In the worst case, Distinguish requires $O(|\mathcal{P}| \times |\mathcal{I}| \times |\mathcal{Q}| \times |\text{Answers}(\mathcal{Q})|)$ CCE calls. However, in practice, Distinguish is efficient because it stops as soon as it finds a single program that is distinguishable from P'. Early in the active learning loop, when the hypothesis space is large, it is easy to find such a program, so this subroutine runs very quickly. After each round of user

procedure CCE(P, Φ, I)
input: P is a program, Φ is feedback, and I is an input.
output: An output set $\tilde{O}$
  $\theta \leftarrow$ ForwardAI(P, Φ, I)
  $\theta \leftarrow$ BackwardAI(P, θ)
  $\Delta \leftarrow$ EvalConsistent(P, θ, Φ)
  return $\{O \mid (O, \varphi) \in \Delta\}$

Fig. 12. Constrained conformal evaluation algorithm.

procedure EvalConsistent(P, θ, I)
input: P is an AST, θ is a constraint specification, and I is an input.
output: Output set $\tilde{O}$
  $n \leftarrow \mathsf{Root}(P)$; $f \leftarrow \mathsf{Label}(n)$; $\Delta \leftarrow \varnothing$
  if $\mathsf{Leaf}(n)$ **then return** $[\![P]\!]_{\theta[n]}(I)$
  for all $n_i \in \mathsf{Children}(n)$ **do**
    $\Delta_i \leftarrow$ EvalConsistent(n_i, θ, I)
  for all $((O_1, \varphi_i), \ldots, (O_k, \varphi_k)) \in \Delta_1 \times \ldots \times \Delta_k$ **do**
    if $\mathsf{SAT}(\bigwedge_i \varphi_i) \wedge [\![f]\!](O_1, \ldots, O_k) \in \gamma(\theta[n])$ **then**
      $\Delta \leftarrow \Delta \cup \{([\![f]\!](O_1, \ldots, O_k), \bigwedge_i \varphi_i)\}$
  return Δ

Fig. 13. Conformal evaluation using abstract values.

procedure ForwardAI(P, Φ, I)
input: P is an AST, Φ is feedback, I is input.
output: Mapping θ from AST nodes to abstract values
  for all $n \in \mathsf{Nodes}(P)$ **do**
    if $\Phi \models \mathsf{Label}(n)(I) = O$ **then**
      $\theta[n] \leftarrow \alpha(O)$
    else $\theta[n] \leftarrow [\![\mathsf{SubProg}(P, n)]\!]^\sharp(I)$
  return θ

Fig. 14. Forward abstract interpretation.

procedure BackwardAI(P, θ)
input: P is an AST, θ is a mapping from AST nodes to abstract values.
output: A strengthened version of θ
  $n \leftarrow \mathsf{Root}(P)$
  $f \leftarrow \mathsf{Label}(n)$
  for all $n_0, \ldots, n_m \in \mathsf{Children}(n)$ **do**
    $\chi \leftarrow [\![f]\!]^{\sharp -i}(\theta[n], \theta[n_0], \ldots, \theta[n_m])$
    $\theta[n_i] \leftarrow \theta[n_i] \sqcap \chi$
    $\theta \leftarrow$ BackwardAI(n_i, θ)
  return θ

Fig. 15. Backward abstract interpretation.

interaction, the hypothesis space shrinks, and although finding a distinguishable program becomes harder, fewer programs remain – hence, the running time continues to be fast.

4.2 Practical Constrained Conformal Evaluation

Several aspects of the ActiveLearning procedure require evaluating *many* programs on *many* inputs using constrained conformal semantics. In particular, in Figures 10 and 11, constrained conformal evaluation is performed using the call to procedure CCE, whose invocations are underlined. Unfortunately, when the prediction sets are large, performing constrained conformal evaluation can be very costly. In this section, we present a CCE algorithm that leverages user-provided input-output examples to make constrained conformal evaluation more efficient in practice.

To understand the key insight behind the CCE procedure, consider the conformal evaluation of a symbolic function f_s: As shown in the Symb rule of Figure 8, an expression $f_s(E_1, \ldots, E_n)$ is evaluated by first evaluating each E_i and then taking the cross product of the resulting sets Δ_i. If each Δ_i has cardinality m, the expression can produce up to m^n outputs, causing an exponential blowup. However, in practice, we find that many of these outputs are *infeasible* given the user feedback provided so far. As an example, consider the following function:

$$\lambda\, x_1, x_2, x_3.(\texttt{toDigit}(x_1) + \texttt{toDigit}(x_2)) \times \texttt{toDigit}(x_3)$$

Suppose the prediction set for x_3 includes $\{2, 4, 7\}$, but the expected output for input I is 9. Since multiplying by an even number produces an even result, the labels 2 and 4 for x_3 must be spurious. Thus, even without explicit user feedback, we can refine x_3's prediction set using this information. This motivates our CCE procedure based on bidirectional abstract interpretation.

As shown in Figure 12, CCE takes three inputs, namely a program P to be evaluated (represented in terms of its AST), the user feedback Φ, and an input I. The result of CCE is a set $\tilde{O}$ such that $\tilde{O} = [\![P]\!]_\Phi(I)$. The algorithm consists of three high-level steps. The first two steps compute a

mapping θ from each node n of the program's AST to an abstract value. The third step, called EvalConsistent (shown in Figure 13), uses the computed mapping θ to refine the prediction sets, with the goal of preventing an exponential blowup in their size as they are propagated through the AST. In particular, after evaluating all the children of a node n (lines 5-6 in Figure 13), EvalConsistent filters evaluation results that are inconsistent with $\theta[n]$ (lines 8–9). Because the filtering operation helps keep the evaluation results of subexpressions in check, EvalConsistent allows for more practical propagation of these output sets.

The CCE procedure computes mapping θ using bidirectional abstract interpretation. The forward direction, summarized in procedure ForwardAI (Figure 14), initializes θ using the user feedback Φ and forward abstract interpretation on input I. Note that, even though we wish to evaluate the program on a *concrete* input I, abstract interpretation is useful for summarizing prediction sets into a *single* abstract value, thereby avoiding potential exponential blowup. In the ForwardAI procedure, we assume access to an abstract semantics $[\![\cdot]\!]^\sharp$ (see line 6) that can be used to compute the abstract output of a program P on a concrete input I.

Given mapping θ initialized via forward abstract interpretation, the CCE procedure invokes BackwardAI to *refine* the abstract values for each node. Starting with the abstract value for the root, BackwardAI (see Figure 15) uses abstract transformers in the inverse direction to compute the abstract values for the children of a construct f. As shown on line 5, we use the notation $[\![\cdot]\!]^{\sharp -i}$ to denote an *inverse abstract transformer* that computes the abstract value for the i'th operand of f, given its output abstract value and the previous abstract values of its children.

Example 4.2. Consider the following program in the hypothesis space:

$$P = \lambda x_1, x_2, x_3.\ \underbrace{\texttt{toDigit}(x_3)}_{a} + \overbrace{(\underbrace{\texttt{toDigit}(x_1)}_{b} \times \underbrace{\texttt{toDigit}(x_2)}_{c})}^{d}$$

Suppose the user provides a new input-output example $\Phi \equiv f_{\text{synth}}(I_1, I_2, I_3) = 16$ where each I_j is an image of a digit, and suppose we have the following prediction sets:

$$[\![\texttt{toDigit}]\!](I_1) = \{1, 2\} \quad [\![\texttt{toDigit}]\!](I_2) = \{2, 4\} \quad [\![\texttt{toDigit}]\!](I_3) = \{6, 7, 8\}$$

Using the ForwardAI procedure, we first compute the following mapping θ in the *interval abstract domain* [19]: $a \mapsto [6, 8]$, $b \mapsto [1, 2]$, $c \mapsto [2, 4]$, $d \mapsto [2, 8]$, and $P \mapsto [16, 16]$. Here, a maps to $[6, 8]$ because the prediction set $\{6, 7, 8\}$ for I_3 is abstracted as $[6, 8]$. The abstract value for P is $[16, 16]$ due to the user-provided output of 16, and d is computed using the forward abstract transformer for multiplication. Next, BackwardAI refines the abstract values. Starting with P and using the inverse semantics for +, we infer $\theta[a] = [8, 14] \sqcap [6, 8] = [8, 8]$, and then $\theta[d] = [8, 8] \sqcap [2, 8] = [8, 8]$. Using the inverse transformer for $\times$, we compute $\theta[b] = [2, 4] \sqcap [1, 2] = [2, 2]$ and $\theta[c] = [4, 4] \sqcap [2, 4] = [4, 4]$. Finally, EvalConsistent evaluates each AST node. For a, the only consistent value in $\{6, 7, 8\}$ is 8, and similarly, 2 and 4 are obtained for b and c. The result of EvalConsistent is the singleton set 16, allowing us to conclude $P \models \Phi$ without exponential blowup during conformal evaluation.

Remark 1. Our CCE strategy fundamentally relies on *both* forward *and* backward reasoning. Without backward reasoning, EvalConsistent would not be able to filter out any values. Conversely, backward reasoning is not possible without forward reasoning, as inverse transformers require the abstract values of the arguments in addition to the output.

Remark 2. Given an input I to which n neural constructs may be applied, the worst case complexity of CCE is $O(n^m)$, where m is the maximum prediction set size. Thus, while our CCE algorithm does not change the *theoretical* worst-case complexity of constrained conformal evaluation, it makes

procedure SelectQuestion($\mathcal{P}, \mathcal{Q}, \Phi$)
input: $\mathcal{P}$ is a hypothesis space, $\mathcal{Q}$ is a question space, and Φ is a feedback formula.
output: Question $q^* \in \mathcal{Q}$ with highest pruning power

$q^* \leftarrow \bot$; $\beta^* \leftarrow 0$ # q^* denotes the best question so far, and β^* is its pruning power
$W \leftarrow$ SortedSet() # Worklist sorted in decreasing order of pruning power
for all $(f, I) \in \mathcal{Q}$ **do** # Worklist initialization
  $\eta \leftarrow \max_{a_i} |\{P \in \mathcal{P} \mid P \models_k (\Phi \wedge f(I) = a_i)\}|$ # Refine HS using BCE
  $\beta \leftarrow (|\mathcal{P}| - \eta)/|\mathcal{P}|$ # β overapproximates the pruning power of (f, I)
  $W \leftarrow W \cup ((f, I), \beta)$
while $W \neq \varnothing$ **do**
  $(q, \beta) \leftarrow W$.first() # Process question with best overapproximate pruning power
  if $\beta^* \geq \beta$ **then** break; # Found question with best pruning power
  $\eta \leftarrow \max_{a_i} |\text{RefineHS}(\mathcal{P}, q, \Phi \wedge f(I) = a_i)|$
  $\beta' \leftarrow (|\mathcal{P}| - \eta)/\mathcal{P}$ # Compute actual pruning power of q
  if $\beta' > \beta^*$ **then** $q^* \leftarrow q$; $\beta^* \leftarrow \beta'$ # Found question with higher pruning power
return q^*

Fig. 16. Question selection algorithm.

a significant difference in practice: our experiments in Section 6 show that the naïve conformal evaluation strategy grows exponentially with respect to prediction set size, whereas our proposed method does not.

4.3 Question Selection

In this section, we describe the SelectQuestion procedure that identifies a question with maximal pruning power. A naive approach would consider all possible questions, compute their pruning powers, and return the best one. Unfortunately, computing the pruning power of a question can be very expensive because it requires refining the hypothesis space and performing constrained conformal evaluation for *every* remaining program. Thus, even with our optimized CCE procedure, this method can be very expensive.

The key idea of our question selection algorithm is to *overapproximate* the pruning power of a question through a new evaluation strategy called *bounded conformal evaluation (BCE)*. The difference between BCE and CCE is similar to the difference between *beam search* and *exhaustive search*. Like beam search limits the beam length to k, BCE restricts output set cardinalities to a fixed k, which acts as a hyper-parameter for our algorithm. BCE approximates the constrained conformal evaluation rules from Figure 8 by modifying the Neural and Symb rules to choose k elements from Δ whenever $|\Delta| > k$. Since BCE is a simple modification of these rules, we omit its formal presentation, using $[\![P]\!]^k_\Phi(I)$ to denote the result of BCE with hyper-parameter k. Unlike CCE, BCE avoids exponential blowup, as all output sets are bounded by k.

Definition 4.3. **(BCE-consistency)** Let P be a program and $\Phi_{\mathcal{F}}$ be user feedback. We say that P is BCE-consistent with $\Phi_{\mathcal{F}}$, denoted $P \models_k \Phi_{\mathcal{F}}$, iff $\forall (f_{\text{synth}}, I, O) \in \mathcal{F}.\ O \in [\![P]\!]^k_{\Phi_{\mathcal{F}}}(I)$.

Our question selection algorithm utilizes the observation that BCE-consistency can be used to over-approximate the pruning power of a question, as stated in the following theorem:

Theorem 4.4. Let $q = (f, I)$ be a question and $\mathcal{P}$ be a program space consistent with Φ. Then:

$$\Pi(q, \mathcal{P}) \leq \frac{|\mathcal{P}| - \max_i |\{P \in \mathcal{P} \mid P \models_k \Phi \wedge f(I) = a_i\}|}{|\mathcal{P}|}$$

We now discuss the SelectQuestion algorithm summarized in Figure 16. This algorithm takes as input a hypothesis space $\mathcal{P}$ consistent with feedback Φ and a set of questions $\mathcal{Q}$, and returns a question $q^* \in \mathcal{Q}$ such that $\forall q \in \mathcal{Q},\ \Pi(q^*, \mathcal{P}) \geq \Pi(q, \mathcal{P})$. To compute q^*, the algorithm maintains a worklist W of pairs (q, β), where q is a question and β is an upper bound on its pruning power. The worklist is initialized in lines 4–7 using BCE-consistency (line 5). The second loop (lines 8–13) processes elements in *decreasing* order of over-approximated pruning power, updating the best question q^* and highest pruning power β^* encountered. Each iteration removes the element with the highest possible pruning power β under BCE semantics. If β^* exceeds β, the search terminates, as β is an upper bound for all remaining questions. Otherwise, the algorithm computes the *actual* pruning power β' of q by calling RefineHS. If β' is larger than β^*, q^* and β^* are updated.

Theorem 4.5. The SelectQuestion procedure returns a question $q^* \in \mathcal{Q}$ with the highest pruning power – i.e., $\forall q \in \mathcal{Q}.\ \Pi(q, \mathcal{P}) \leq \Pi(q^*, \mathcal{P})$.

Remark 1. Any evaluation strategy that over-approximates pruning power (i.e., under-approximates $\mathcal{P} \downarrow \Phi$) could be used instead of BCE without affecting the optimality of SelectQuestion.

Remark 2. Since SelectQuestion procedure performs RefineHS at most once for every (question, answer) pair, it can require up to $O(|\mathcal{P}|\times|\mathcal{I}|\times|\mathcal{Q}|\times|\mathsf{Answers}(\mathcal{Q})|)$ calls to CCE. While the theoretical worst-case complexity of Distinguish is the same as that of SelectQuestion, empirically, we find that SelectQuestion dominates the runtime of ActiveLearning (see Section 6.6).

5 Implementation

We have implemented the proposed active learning technique as a new tool called SmartLabel written in Python. SmartLabel can be customized for different application domains by (1) instantiating the meta-DSL from Figure 7 with a specific DSL, (2) defining a suitable abstract domain, along with appropriate forward and backward abstract transformers, and (3) supplying a synthesis engine that can be used to construct the initial hypothesis space for that domain.

Sampling. Following prior work [39], we compute the pruning power of each question on a sampled subset of the hypothesis space. As shown in [39], this sampling procedure assures that the selected query has pruning power ε-close to that of the optimal query with bounded probability.

BCE generalization. Recall that our presentation of the question selection algorithm uses BCE to over-approximate the pruning power of a question. SmartLabel implements a generalized version of BCE that samples $\mathsf{min}(k, n \times k')$ elements from the prediction set, where k is a positive integer, k' is a real-valued percentage in the range $(0, 1)$, and n is the size of the prediction set. This generalized BCE strategy allows sampling some percentage k' of the prediction set up to some limit k. All hyper-parameters used in our implementation, including k and k', are chosen for each domain based on a validation set.

Computing prediction sets for neural functions. Our evaluation technique utilizes conformal prediction, a technique for augmenting machine learning methods with prediction sets that guarantee coverage of the true labels with some user-specified probability. Given any model $f : \mathcal{X} \to \mathcal{Y}$, the goal is to learn a model $\tilde{f} : X \to 2^{\mathcal{Y}}$ that produces sets of labels that include the true label with high probability. Critically, this guarantee should be achieved regardless of how well or poorly the initial model f performs. We assume the existence of a *calibration dataset* $Z \in \mathcal{D}^n$ with which to train $\tilde{f}$, and a *non-conformity score* $s_f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, which is a heuristic measure of how "close" the prediction $f(x)$ is to any label y. Intuitively, a smaller score $s_f(x, y)$ means that y has a higher likelihood of being the true label of x. The basic idea of conformal prediction is to consider the

collection of prediction-set functions $\{\tilde{f}_\tau\}_{\tau\in\mathbb{R}}$ parameterized by a thresholding score-value τ:

$$\tilde{f}_\tau(x) = \{y \in \mathcal{Y} : s_f(x, y) \leq \tau\}$$

The calibration dataset Z gives an empirical distribution over the non-conformity scores induced by $\mathcal{D}$, and using this, a thresholding value $\tilde{\tau}$ can be chosen to achieve a guarantee of the form:

$$\mathbb{P}_{Z\sim\mathcal{D}^n,(x,y)\sim\mathcal{D}}[y \in \tilde{f}_{\tilde{\tau}}(x)] \geq 1 - \delta$$

Here, $\mathcal{D}$ is the fixed distribution from which data is drawn, and $\delta \in (0, 1]$ is the miscoverage rate. Although conformal prediction makes no assumptions on how s_f is defined, a well-constructed non-conformity score often results in smaller prediction sets.

6 Evaluation

In this section, we describe the results of our experimental evaluation, which aims to answer the following research questions:

- **RQ1:** Can our active learning approach identify the ground truth program, and how do the results compare against prior work on active learning?
- **RQ2:** How many rounds of user interaction are required, and how does this compare against alternative question selection approaches?
- **RQ3:** How important are our key algorithmic ingredients for the runtime and scalability of our approach?
- **RQ4:** Which components of SmartLabel dominate its runtime in practice?

6.1 Application Domains and Benchmarks

To answer our research questions, we instantiate SmartLabel for three application domains from prior work: batch image editing, image search, and visual arithmetic. In what follows, we provide a brief description of each of these domains and their DSLs. The neural components of each DSL are denoted in **bold**. We refer the interested reader to Appendix C for more details about the abstract domains and corresponding abstract transformers.

6.1.1 Batch Image Editing. Our first application domain, ImageEdit, consists of the neurosymbolic image editing DSL (see Figure 17) and batch image editing tasks considered in recent work [7]. This DSL uses neural components (e.g. image segmentation, object classification, facial recognition, etc.) to construct a symbolic representation of an image, and applies manipulations (e.g. crop, blur, etc.) to *parts* of the image that match some criterion. For this application, we use the same abstract domain proposed in [7] but additionally define backward abstract transformers for each DSL construct. We evaluate on the *same tasks* from [7], excluding 13 text-related benchmarks due to the lack of conformal predictors for the underlying neural networks. The input spaces for these tasks consist of the same image datasets considered in [7] (e.g. an album of 359 wedding photos collected from Flickr). Note that neural components in this domain perform binary classification; hence, the maximum size of each prediction set for this domain is 2.

$$\begin{array}{rcl} P & := & \{E \to A, \cdots, E \to A\} \\ A & := & \mathsf{Blur} \mid \mathsf{Brighten} \mid \mathsf{Crop} \mid \cdots \\ E & := & \mathsf{All} \mid \textbf{Is}(\varphi) \mid \mathsf{Complement}(E) \\ & \mid & \mathsf{Union}_N(E_1, \cdots, E_N) \\ & \mid & \mathsf{Intersect}_N(E_1, \cdots, E_N) \\ & \mid & \mathsf{Find}(E, \varphi, f) \mid \mathsf{Filter}(E, \varphi) \\ \varphi & := & \textbf{Object}(O) \mid \textbf{Smiling} \mid \cdots \cdots \\ f & := & \mathsf{GetLeft} \mid \mathsf{GetRight} \mid \mathsf{GetAbove} \mid \cdots \end{array}$$

Fig. 17. ImageEdit DSL.

Notably, our experimental setup differs from that of [7], wherein the reported results rely on repeated synthesis attempts guided by an external oracle. This oracle checks whether the synthesized program is observationally equivalent to the ground-truth program and provides additional examples as needed. Our evaluation setting does not assume access to such an oracle.

6.1.2 Image Search and Visual Concept Discovery. Our second domain, ImageSearch, involves tasks that require finding all images in a corpus that exhibit a certain property (e.g., containing both a dog and a cat) or that involve a "visual concept" (e.g., guitar player, still life photo). Such tasks have been considered both in the neurosymbolic synthesis literature [8] as well as machine learning and computer vision literature [56, 72]. We consider the DSL (see Figure 18) and tasks from [8] as well as a set of visual concept discovery tasks [56]. The input space for tasks in this domain consists of the same real-world images as the ImageEdit domain. Furthermore, the abstract domain is the same as ImageEdit, with transformers adapted to the image search DSL. In this domain, neural networks also perform binary classification; hence the maximum prediction set size is also 2.

$$
\begin{array}{rl}
E := & r(t_1, \ldots, t_n) \\
\mid & E \rightarrow E \mid E \wedge E \mid E \vee E \\
\mid & \neg E \mid \exists x.E \mid \forall x.E \\
r := & \textbf{HasType} \mid \textbf{HasEmotion} \\
\mid & \textbf{HasProperty} \mid \text{HasRelation} \\
t := & x \mid c
\end{array}
$$

Fig. 18. ImageSearch DSL.

6.1.3 Visual Arithmetic. As a third application domain, we consider *visual arithmetic* tasks (e.g., adding or multiplying hand-written digits) from prior work on neurosymbolic programming [10, 47, 48, 54]. These tasks involves performing computations over a list of digit images using functional combinators such as `map`, `fold`, and `filter`, using the DSL in Figure 19. The input space for this application domain consists of lists of digits from the SVHN dataset [55], which is a more challenging version of the MNIST dataset [45] comprised of images of digits in natural scene images.

$$
\begin{array}{rl}
P := & \lambda\, l.\, E \\
E := & l \mid c \in \mathbb{N} \mid \texttt{fold}\ f\ c\ E \\
\mid & \texttt{map}\ g\ E \mid \texttt{filter}\ h\ E \\
f := & \texttt{sum} \mid \texttt{max} \\
\mid & \texttt{product} \mid \texttt{inc} \\
g := & \texttt{curry}\ f\ c \mid \textbf{toDigit} \\
h := & \lambda x.x \lhd c
\end{array}
$$

Fig. 19. VisArith DSL.

Since prior work does not introduce an abstract domain for this application, we consider a standard interval abstraction for each list element. Unlike the first two domains where all neural components perform binary classification, the neural constructs in this DSL perform digit classification; hence, the maximum size of prediction sets for this domain is 9.

To give the reader some intuition about the tasks considered in our evaluation, Table 1 shows two representative tasks for each domain, along with their corresponding ground truth programs.

6.2 Experimental Set-up and Methodology

We evaluate SmartLabel (as well as all baselines and ablations) using the following methodology: First, we sample two inputs from the input space and supply the corresponding output, giving us a set Z of initial input-output examples. Then, we compute $\mathcal{P} = \{P \mid \forall (I, O) \in Z.\, O \in [\![P]\!](I)\}$ by enumerating all programs up to a fixed AST size of 20.

We then use SmartLabel and all baselines to perform active learning on $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{I}$ and Z, using the ground truth programs and labels to emulate a user responding to queries. To account for variability from the random IO examples, we repeat all experiments five times, each with a different seed, and report the mean and standard deviation of the outcomes. All experiments are run on a 2022 MacBook Pro with an 8-core m2 processor and 8 GB RAM, with a timeout limit of 600s.

6.2.1 Benchmark statistics. As summarized in Table 2, we evaluate SmartLabel on 112 tasks. On average, the initial hypothesis space contains around 1510 programs, the question space contains around 800, and each question has approximately 9.8 possible answers. Recall from Section 3.4 that the question space $\mathcal{Q}$ comprises (1) questions about the ground-truth output for each input, and (2) questions about the ground-truth label for each (neural component, input) pair. Hence, $\mathcal{Q}$ is fixed for a given task. As expected, prediction set sizes are larger in the VisArith domain, since the prediction space includes 9 labels rather than 2 for binary classification. However, the question

Table 1. Representative tasks in the ImageEdit, ImageSearch, and VisArith domains.

Domain	Task Description	Ground Truth Program
ImageEdit	Crop out everyone who is not smiling or has their eyes closed.	{Intersect(**Is**(**Object**(face)), Complement(Intersect(**Is**(**Smiling**), **Is**(**EyesOpen**)))) → CropOut}
ImageEdit	Blur the faces of people who are not playing guitar.	{Intersect(**Is**(**Object**(face)), Complement(Find((**Is**(**Object**(guitar)), **Is**(**Object**(face)), GetAbove)))) → Blur}
ImageSearch	Find all images that contain a cyclist wearing a helmet.	$\exists x.\exists y.\exists z.$**HasType**$(x,$ bicycle$)\wedge$**HasType**$(y,$ person$)$ $\wedge$**HasType**$(z,$ helmet$)\wedge$ HasRelation$(x, y,$ below$)$ $\wedge$HasRelation$(y, z,$ below$)$
ImageSearch	Find all images that do not contain any people.	$\forall x.\neg$**HasType**$(x,$ person$)$
VisArith	Compute the sum of a list after doubling all items.	`fold plus 0 (map (curry prod 2) (map` **`toDigit`** l`))`
VisArith	Compute the maximum list item that is smaller than the head of the list.	`fold max 0 (filter (curry` $\lambda.x <$ `(head` l`))(map` **`toDigit`** l`))`

Table 2. Details about the (1) number of tasks, average sizes of (2) initial hypothesis space, (3) final hypothesis space, (4) input space, (5) question space, (6) answer space per question, (7) prediction set, and (8) average percentage of inputs with mispredictions.

Domain	# Tasks	|Initial HS|	|Final HS|	$\lvert\mathcal{I}\rvert$	$\lvert Q\rvert$	$\lvert$Possible$(q)\rvert$	|Pred. Set|	% Inputs w/ Mispred.
ImageEdit	37	1731.0	5.7	271.7	1050.8	11.3	1.2	53.0%
ImageSearch	25	593.8	3.7	257.4	1135.1	12.7	1.1	59.0%
VisArith	50	1813.7	32.1	100.0	444.0	7.3	2.5	13.0%
Overall	112	1514.1	17.1	191.9	797.4	9.8	1.8	30.0%

space is significantly larger for ImageEdit and ImageSearch due to (1) the larger input space, and (2) the greater number of neural components in their DSLs.

To quantify the quality of the neural components used in our experiments, we also report the average percentage of inputs with at least one misprediction. In the ImageEdit and ImageSearch domains, slightly over half of inputs contain a neural misprediction, on average. In these domains, a misprediction may be an undetected object, an erroneous object detection, or a misclassified attribute. In VisArith, 13.0% of inputs contain a neural misprediction, on average. In this domain, a misprediction is a misclassified digit. These statistics demonstrate the systemic uncertainity of neural mispredictions across different domains and models.

Together, these three domains span diverse characteristics, forming a comprehensive test bed for evaluating our approach.

6.3 Comparison Against Existing Active Learning Techniques

To answer our first research question, we compare SmartLabel against two existing active learning techniques proposed in prior work for traditional program synthesis:

(1) **SampleSy** [39], an active learning procedure whose question selection algorithm uses an objective function similar to that of SmartLabel. In each round of interaction, SampleSy selects the question whose worst answer eliminates the largest number of programs.
(2) **LearnSy** [38], an active learning procedure that selects the input maximizing the approximate number of pairs of programs whose outputs differ.

Importantly, both of these baselines evaluate program consistency using the DSL's standard semantics – that is, they treat the neural prediction as the ground truth and produce a single output when executing a neurosymbolic program on a given input. While prior work proposes synthesis techniques for similar application domains (e.g., PhotoScout [8] solves image search tasks using a neurosymbolic DSL), these techniques do not explicitly address the neurosymbolic active learning problem. Hence, such approaches are not baselines for evaluating the contributions of our work.

The results of this experiment are summarized in the "# Benchmarks Solved" column of Table 3. Here, a benchmark is considered solved if the synthesizer returns a program that is observationally equivalent to the ground truth program under the ground truth semantics. For SmartLabel, failure to return the ground truth program can occur for two reasons: (1) timing out, and (2) incompleteness of conformal prediction. Although conformal prediction includes the ground truth label in the prediction set with *high probability*, there is a small but non-zero chance that it will be excluded. In the baselines, failure can also occur due to timing out or due to mispredictions by the neural classifiers. Unlike SmartLabel, the baselines do not use conformal semantics, making it more likely for the ground truth program to be pruned from the search space.

As shown in Table 3, SmartLabel significantly outperforms the baselines in terms of its ability to converge to the ground truth program: SmartLabel finds the ground truth program for over 98% of the benchmarks, whereas SampleSy and LearnSy output the correct program for 65% and 58% of benchmarks, respectively.

To gain insight about the failures of SmartLabel, we manually inspected the 9 cases out of 560 runs where SmartLabel fails to output the intended program. Out of these 9 cases, 2 are due to timeouts. For the remaining 7 cases, failure is caused by the intended program's output set not containing the ground truth for a particular input. For all of these 7 benchmarks, we confirmed that our method does converge to the intended program if we manually increase the confidence threshold used for conformal prediction, thereby making conformal prediction more conservative (albeit at the cost of making user interaction times longer). We also performed a similar investigation for the failure cases of the two baselines: SampleSy fails on 196 out of the 560 runs, whereas LearnSy fails on 235. There are no timeouts for either of the baselines; hence, all failures are caused by using standard semantics instead of conformal semantics. These results underscore the need for conformal semantics in the context of neurosymbolic active learning.

Table 3. Experimental results comparing the number of benchmarks solved by SmartLabel versus two baseline active learning techniques.

Domain	# Benchmarks Solved		
	Ours	SampleSy	LearnSy
ImageEdit	95.7% ± 2.4%	70.3% ± 5.1%	58.9% ± 14.6%
ImageSearch	99.2% ± 1.8%	72.8% ± 8.8%	55.2% ± 12.4%
VisArith	100% ± 0%	57.2% ± 5.4%	58.8% ± 2.3%
Overall	98.4 ± 0.7%	65.0% ± 5.0%	58.0% ± 6.0%

Result for RQ1: SmartLabel's use of constrained conformal semantics leads to significantly higher success rates in finding the desired program compared to the baselines.

6.4 Evaluation of Rounds of User Interaction

To answer our second research question, we evaluate the number of rounds of user interaction required to converge to the ground truth program. To further evaluate our question selection strategy, we also compare the worst-case pruning power objective against two alternative question selection strategies:

- **Expected:** This variant selects questions that maximize the *expected* number of programs prunes, rather than the worst-case. Specifically, for each candidate question $q = (f, I)$, we compute the prediction set $\{a_1, \ldots, a_n\}$ for $f(I)$, along with the associated confidence scores from the neural

predictor. These scores are normalized to form a probability distribution $\{p_1, \ldots, p_n\}$ over possible answers. The expected pruning power of q is then given by: $\text{EPP}(q) = \sum_{i=1}^{n} p_i \cdot \Pi(q, P \mid a_i)$. Here $\Pi(q, P \mid a_i)$ denotes the fraction of programs pruned from the hypothesis space assuming the user's answer is a_i. This strategy favors questions that eliminate many incorrect programs *in expectation*.

- **Random:** Rather than optimizing a certain objective during question selection, this alternative question selector randomly samples a question from the question space.

Table 4. Experimental results comparing the average number of rounds of user interaction and average time per round of interaction taken by SmartLabel versus two alternative question selection strategies. Our proposed question selection strategy, which uses a worst-case pruning power optimization objective, is denoted by the WorstCase column. The Expected column denotes a strategy that uses an *expected* pruning power optimization objective, and the Random column denotes a strategy that samples a question.

Domain	Average # Rounds of Interaction			Average Time per Round of Interaction (s)		
	WorstCase	Expected	Random	WorstCase	Expected	Random
ImageEdit	5.0 ± 1.7	6.3 ± 2.0	113.4 ± 29.5	7.3 ± 2.8	48.5 ± 14.1	0.5 ± 0.2
ImageSearch	6.4 ± 1.6	6.5 ± 2.2	145.8 ± 54.6	4.9 ± 1.8	46.9 ± 12.8	0.1 ± 0.0
VisArith	4.0 ± 0.1	9.8 ± 0.2	53.5 ± 2.2	4.4 ± 0.2	11.2 ± 0.3	0.3 ± 0.0
Overall	4.9 ± 0.5	7.9 ± 0.7	93.9 ± 17.5	5.5 ± 1.4	26.8 ± 3.6	0.3 ± 0.0

We emphasize that both of these alternative strategies rely on the machinery introduced in this paper (e.g., constrained conformal evaluation) to guarantee convergence to the ground truth program. Thus, the differences in performance can be attributed entirely to the choice of question selection strategy.

The results of this evaluation are presented in Table 4. When using our proposed *worst-case pruning power* optimization objective from Definition 3.11, our method takes an average of 4.9 rounds of user interaction. In other words, SmartLabel can disambiguate between the initial 1514 programs in the hypothesis space by asking fewer than 5 questions, on average. As expected, the Random strategy performs significantly worse, requiring an average of 93.9 rounds of user interaction – nearly 20 times more than our WorstCase strategy. While the Expected strategy performs better than Random, it still requires 7.9 rounds on average. This result is somewhat surprising, given that Expected leverages prediction probabilities to guide question selection. In practice, however, we found that neural networks can sometimes be confidently wrong about their predictions, which can causes the expected pruning power to prioritize misleading questions.

Additionally, Table 4 shows the average time per user interaction round for each strategy. The Random strategy is extremely fast, since it avoids any computation during question selection and simply samples a query uniformly at random. However, despite its low per-round overhead, Random still takes significantly longer overall. Even assuming an optimistic estimate of just 3 seconds for the user to answer each query, Random would take over 300 seconds to converge – more than 7 times longer than the total active learning time required by the WorstCase strategy.

In contrast, the Expected strategy incurs a much higher cost per round compared to WorstCase, as it requires propagating the predicted probabilities of neural components during program evaluation. Overall, these results empirically demonstrate that optimizing for worst-case pruning power yields a favorable trade-off between the *number* of rounds of user interaction and the *time* to compute a question per round.

Result for RQ2: The optimization objective based on worst-case pruning power yields the best trade-off between number of rounds and computation time compared to two alternative question selection strategies.

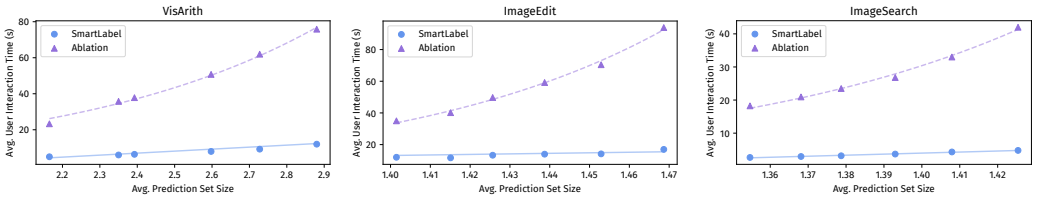


Fig. 20. User interaction time as prediction set size increases for each domain.

6.5 Ablation Studies

In this section, we aim to evaluate the impact of our two key algorithmic optimizations through ablation studies. For this evaluation, we consider the following ablations of SmartLabel:

- **No-AbsInt:** This variant disables our proposed bidirection abstract interpretation method in the CCE algorithm. In particular, this ablation does *not* call ForwardAI and BackwardAI in the CCE procedure; it also omits the consistency check in lines 7–8 of the EvalConsistent method.
- **No-BCE:** This ablation does not utilize bounded conformal evaluation to derive an upper bound on the pruning power of each question. In more detail, it omits the initial for loop in lines 4–7 of the SelectQuestion procedure; it also omits the pruning test at line 13.

6.5.1 Runtime. Table 5 compares the runtime of SmartLabel against those of the two ablations. Since these ablations do not impact the number of rounds of user interaction, we only compare the time it takes to generate a question. Across all domains, SmartLabel takes an average of 5.5 seconds per round, while No-AbsInt and No-BCE take roughly 2× and 6× longer, respectively. This difference highlights the impact of these key optimizations on our method's practicality.

6.5.2 Scalability. As discussed earlier in Section 4, our algorithm scales linearly with respect to various parameters like input and question space size; however, its worst-case time complexity is exponential with respect to *prediction set size*. Here, we evaluate the effectiveness of our proposed algorithmic optimizations (namely, CCE with bidirectional reasoning and BCE for question selection) in mitigating this worst-case blowup in practice. To perform this experiment, we increase prediction set sizes by varying the parameters of conformal prediction (see Section 5 for more details). Intuitively, by varying this parameter, we make conformal prediction more conservative, leading to larger prediction set sizes.

Table 5. Experimental results comparing the average time per round of interaction taken by SmartLabel versus two ablations.

Domain	Average Time per Round of Interaction (s)		
	Ours	No-BCE	No-AbsInt
ImageEdit	7.3 ± 2.8	42.1 ± 8.5	16.9 ± 5.1
ImageSearch	4.9 ± 1.8	41.9 ± 8.1	8.9 ± 2.2
VisArith	4.4 ± 0.2	21.5 ± 0.5	8.5 ± 0.5
Overall	5.5 ± 1.4	33.3 ± 3.8	11.2 ± 2.3

The result of this evaluation is presented in Figure 20, which shows how the average user interaction time increases with respect to prediction set size in each domain. In these plots, the x-axis represents average prediction set size across the entire input space, and the y-axis represents average user interaction. The blue dots show the results for SmartLabel, while the purple triangles represent the results of an ablation that uses neither bidirectional reasoning nor BCE (i.e. it is a combination of both No-AbsInt and No-BCE). As demonstrated by the divergence between the two sets of dots, our algorithmic optimizations are indeed effective at preventing an exponential blowup. In particular, the blue line fits the SmartLabel results roughly linearly (best fit $ax + b$, $R^2 = 0.97$ for VisArith, $R^2 = 0.94$ for ImageEdit, and $R^2 = 0.98$ for ImageSearch), while the purple dashed line fits the ablation results exponentially (best fit ab^x, $R^2 = 0.99$ for VisArith, $R^2 = 0.99$ for ImageEdit, and $R^2 = 0.99$ for ImageSearch).

Result for RQ3: Our key algorithmic ingredients have a significant positive impact on both average runtime per user interaction round as well as on the algorithm's scalability with respect to prediction set size.

6.6 Evaluation of the Impact of Active Learning Components on Runtime

Recall that our active learning procedure involves three core components: (1) RefineHS, which prunes inconsistent programs using CCE; (2) Distinguish, which checks whether further disambiguation is necessary; and (3) SelectQuestion, which identifies the next query by maximizing an objective. While Section 4 analyzes the theoretical worst-case complexity of these components, an interesting *empirical* question is where SmartLabel spends the majority of its runtime in practice. Figure 21 shows how the three main components of our algorithm contribute to user interaction time. As shown in this figure, most of the runtime (69.5%) is consumed by the SelectQuestion procedure, followed by RefineHS (26.4%), and only 4.1% is spent on Distinguish. Despite its worst-case complexity, the practical efficiency of Distinguish can be explained by two factors. First, in early rounds of active learning, there are many distinguishable programs in the hypothesis space, and Distinguish only needs to find a single distinguishing pair to conclude that learning should continue – allowing it to terminate quickly. Second, in later rounds, although distinguishing programs becomes harder, the hypothesis space is much smaller – reducing the overall search burden.

Result for RQ4: The runtime of active learning is dominated by SelectQuestion, followed by RefineHS. In contrast, Distinguish is very fast in practice despites its worst-case theoretical complexity.

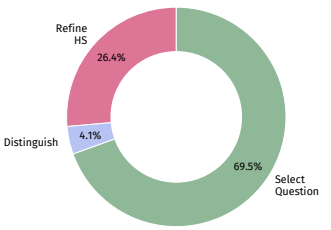


Fig. 21. Breakdown of active learning runtime.

7 Related Work

Active learning for program synthesis. Active learning [11, 12, 20, 62, 64, 66] is a type of machine learning approach where the algorithm selectively chooses the data from which it learns. Instead of using a large, randomly selected dataset, the model actively selects specific data points to label, with the goal of minimizing the number of queries to the human. In recent years, there has been significant interest in active learning techniques for *interactive program synthesis*. The goal of these techniques is to query the user until there is no remaining ambiguity in the specification. A variety of techniques [25, 37, 44, 51, 57] select a question that distinguishes at least two programs in the program space. However, these techniques do not address the *optimal question selection* problem in the PBE setting, where the goal is to minimize the rounds of user interaction. Similarly to SmartLabel, *SampleSy* and *EpsSy* [39] both employ a greedy question selection algorithm that selects the question whose worst-case answer will result in the best outcome. The followup work, *LearnSy* [38], learns a model that predicts the likelihood that two programs will have the same output on a question. However, these prior algorithms are designed to synthesize programs in *purely symbolic* languages, and do not take into account the discrepancy between the ground truth and evaluation semantics of neural constructs. As a result, they do not offer any correctness or completeness guarantees in the neurosymbolic setting.

Neurosymbolic program synthesis. There is much recent work in applying program synthesis techniques to neurosymbolic DSLs. In the image processing domain, various works [7, 8, 22, 31, 36, 41, 73] synthesize programs that combine symbolic operators with neural perception modules. In the

data extraction and manipulation domain, several recent works [15–17, 40, 79] synthesize programs that utilize large language models to reason semantically about data. Other works [27, 68, 76] provide more general frameworks for synthesizing programs that compose neural and symbolic operators. A common limitation among these works is that the synthesized program is only guaranteed to match the user's specification under the assumption that the program's underlying neural components are always correct. Our work proposes a framework for neurosymbolic program synthesis that is robust to mispredictions of neural models.

Conformal prediction. In recent years, conformal prediction [2, 67] has become a popular paradigm for uncertainty quantification of black-box learning models, finding use in a variety of learning tasks across different domains [1]. In the traditional setting, one limiting assumption must be made about the test data in order to achieve the desired coverage guarantees – namely that it is exchangeable with the calibration data. Much work in the area has thus focused on adapting conformal techniques to more general learning settings, such as distribution shift [28, 59, 74], time-series [18, 86], and online settings [9, 29]. A very recent (unpublished) work [60] aims to extend conformal prediction (parameterized by a user-specified confidence bound) from individual neural components to neurosymbolic programs, using a calibration set, a scoring function (for measuring non-conformity), and a user-specified confidence bound. Since the prediction sets obtained this way can be quite imprecise, they also use abstract interpretation to tighten them. Our work builds on this effort in defining conformal semantics for neurosymbolic languages, but allows the user to refine the conformal semantics through user feedback. Additionally, the main focus of our paper is neurosymbolic active learning, which is not addressed in prior work.

Abstract interpretation in program synthesis. Prior work has proposed pruning techniques for program synthesis that leverage abstract interpretation. Many such approaches employ abstract reasoning in only the forward direction [23, 32, 52, 69, 70, 81, 83], or only the backward direction [58]. Other works utilize abstraction refinement to generate a program that matches a set of IO examples under an abstract semantics, and then iteratively refine the semantics if the generated program is spurious [30, 77, 83, 84]. Several works leverage both forward and backward reasoning in program synthesis. In particular, [49] and [53] use bidirectional abstract reasoning to efficiently prune infeasible partial programs in their respective domains (automated transpilation and LLVM superoptimization). Yoon et al. [87] propose a general framework for inductive synthesis that uses iterative forward-backward abstract interpretation to infer constraints on partial programs. In contrast to all of these approaches, our method uses forward and backward abstract interpretation to speed up conformal evaluation, which is not addressed in prior work.

Program synthesis with noisy data. Various works contend with the problem of synthesizing a program from noisy IO examples. In general, neural program synthesizers are more robust to noise than symbolic synthesizers. For instance, RobustFill [21] uses an RNN-based approach to generate string transformations programs from IO examples with noise (e.g. output strings with typos). However, purely neural synthesis approaches do not offer correctness guarantees. Handa and Rinard [34] proposes a general framework for inductive synthesis over noisy data that finds a program that minimizes a user-defined cost function. This cost function quantifies how often the program differs from the IO examples. Its followup, [33], bounds the error rate of this framework with respect to a formalized noise source. Raychev et al. similarly propose a synthesis technique for noisy data that outputs a "best fit" program for a cost function; this approach also bounds the error rate in the case that the amount of noise in the dataset is bounded [61]. In contrast to these works, SmartLabel guarantees that the ground truth program is synthesized under the assumption that prediction sets contain the ground truth label for neural components.

8 Conclusion

In this paper, we defined the neurosymbolic active learning problem and proposed the first technique for solving it. Our approach uses a new evaluation strategy called *constrained conformal evaluation (CCE)* to account for inaccuracies in neural network predictions. We have evaluated our approach – implemented in a new tool called SmartLabel – on 112 neurosymbolic program synthesis benchmarks across three domains. Our experimental results point to three key findings: First, SmartLabel is able to identify the desired program for over 98% of the benchmarks, whereas prior techniques for active learning can only converge to the ground truth for at most 65% of the benchmarks. Second, the questions identified by SmartLabel result in effective user interaction, requiring 19× fewer rounds of user interaction compared to a random question selection strategy. Finally, our ablation studies show the importance of our key algorithmic optimizations, both in terms of reducing user interaction time and scaling with respect to prediction set sizes.

9 Future Work

While our evaluation focuses on domain-specific languages, SmartLabel is not inherently limited to this setting. A promising direction for future work is to apply our approach to more expressive, general-purpose languages. The core components of our method (namely, active learning via constrained conformal evaluation) remain applicable so long as the user can provide an abstract interpreter for their target language. Prior work has implemented abstract transformers for a range of expressive languages [3, 26, 42].

General-purpose languages typically have larger program spaces, and may require different specification formats. One natural approach to scaling to such languages is to leverage large language models (LLMs). LLM-based techniques have recently shown promise in program synthesis tasks by generating candidate programs from natural language prompts [4, 8]. While LLMs are often noisy and imprecise, they can be used to produce a diverse candidate set that serves as the initial hypothesis space for SmartLabel. Our method is well-suited to this setting, as the CCE algorithm is designed to resolve uncertainty and eliminate incorrect candidates through user interaction.

Another avenue for future work is to explore more compact or symbolic hypothesis space representations, such as version-space algebras (VSAs), tree automata, or partial program sketches. While our experiments use enumerative synthesis to generate the initial hypothesis space, our active learning approach is compatible with any hypothesis space representation that (1) guarantees inclusion of the ground-truth program, and (2) supports pruning of inconsistent candidate programs. Alternative hypothesis space representations may enable more efficient synthesis in domains where enumerative search is infeasible.

More broadly, the problem our method addresses (i.e., ambiguity caused by uncertain neural predictions) applies to a wide range of neurosymbolic systems, not just the ones considered in this paper. While our evaluation focuses on systems where neural and symbolic components are tightly integrated (e.g., neural functions embedded in a DSL), similar challenges arise in more loosely coupled architectures. For example, some recent agentic systems use a large language model to orchestrate a set of symbolic tools or modules, calling different components based on a high-level plan [13, 82, 85]. These systems typically treat neural components as black boxes and lack systematic ways to detect or resolve uncertainty. We believe that extending our conformal evaluation and querying strategy to such agent-based architectures could improve their reliability – especially in cases where neural predictions might silently lead to incorrect results.

10 Data-Availability Statement

The artifact for this paper is available on Zenodo [6].

Acknowledgments

We would like to thank the members of the UTOPIA group, and the anonymous reviewers, for their help and feedback on this paper. This work was conducted in a research group supported by NSF awards CCF-1762299, CCF-1918889, CNS-1908304, CCF-1901376, CNS-2120696, CCF-2210831, and CCF-2319471, CCF-2422130, CCF-2403211 as well as a DARPA award under agreement HR00112590133.

References

[1] Anastasios Angelopoulos, Stephen Bates, Jitendra Malik, and Michael I Jordan. 2020. Uncertainty sets for image classifiers using conformal prediction. *arXiv preprint arXiv:2009.14193* (2020).

[2] Anastasios N Angelopoulos, Stephen Bates, et al. 2023. Conformal prediction: A gentle introduction. *Foundations and Trends® in Machine Learning* 16, 4 (2023), 494–591.

[3] Vincenzo Arceri and Isabella Mastroeni. 2021. Analyzing dynamic code: a sound abstract interpreter for evil eval. *ACM Transactions on Privacy and Security (TOPS)* 24, 2 (2021), 1–38.

[4] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732* (2021).

[5] Vineeth Balasubramanian, Shen-Shyang Ho, and Vladimir Vovk. 2014. *Conformal prediction for reliable machine learning: theory, adaptations and applications*. Newnes.

[6] Celeste Barnaby, Qiaochu Chen, Ramya Ramalingam, Osbert Bastani, and Isil Dillig. 2025. *Artifact for "Active Learning for Neurosymbolic Program Synthesis"*. doi:10.5281/zenodo.16915436

[7] Celeste Barnaby, Qiaochu Chen, Roopsha Samanta, and Işıl Dillig. 2023. ImageEye: Batch Image Processing Using Program Synthesis. *Proc. ACM Program. Lang.* 7, PLDI, Article 134 (jun 2023), 26 pages. doi:10.1145/3591248

[8] Celeste Barnaby, Qiaochu Chen, Chenglong Wang, and Isil Dillig. 2024. PhotoScout: Synthesis-Powered Multi-Modal Image Search. *arXiv preprint arXiv:2401.10464* (2024).

[9] Osbert Bastani, Varun Gupta, Christopher Jung, Georgy Noarov, Ramya Ramalingam, and Aaron Roth. 2022. Practical adversarial multivalid conformal prediction. *Advances in Neural Information Processing Systems* 35 (2022), 29362–29373.

[10] Samuele Bortolotti, Emanuele Marconato, Tommaso Carraro, Paolo Morettin, Emile van Krieken, Antonio Vergari, Stefano Teso, and Andrea Passerini. 2024. A Neuro-Symbolic Benchmark Suite for Concept Quality and Reasoning Shortcuts. *arXiv preprint arXiv:2406.10368* (2024).

[11] Nader H Bshouty. 1995. Exact learning boolean functions via the monotone theory. *Information and Computation* 123, 1 (1995), 146–153.

[12] Nader H Bshouty, Richard Cleve, Sampath Kannan, and Christino Tamon. 1994. Oracles and queries that are sufficient for exact learning. In *Proceedings of the seventh annual conference on Computational learning theory*. 130–139.

[13] Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2023. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201* (2023).

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

[15] Qiaochu Chen, Arko Banerjee, Çağatay Demiralp, Greg Durrett, and Işıl Dillig. 2023. Data Extraction via Semantic Regular Expression Synthesis. *Proceedings of the ACM on Programming Languages* 7, OOPSLA2 (2023), 1848–1877.

[16] Qiaochu Chen, Aaron Lamoreaux, Xinyu Wang, Greg Durrett, Osbert Bastani, and Isil Dillig. 2021. Web question answering with neurosymbolic program synthesis. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 328–343.

[17] Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, et al. 2022. Binding language models in symbolic languages. *arXiv preprint arXiv:2210.02875* (2022).

[18] Victor Chernozhukov, Kaspar Wüthrich, and Zhu Yinchu. 2018. Exact and Robust Conformal Inference Methods for Predictive Machine Learning with Dependent Data. In *Conference On Learning Theory*. 732–749.

[19] Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. 238–252.

[20] Sanjoy Dasgupta. 2004. Analysis of a greedy active learning strategy. *Advances in neural information processing systems* 17 (2004).

[21] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. 2017. Robustfill: Neural program learning under noisy i/o. In *International conference on machine learning*. PMLR, 990–998.

[22] Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. 2018. Learning to infer graphics programs from hand-drawn images. *Advances in neural information processing systems* 31 (2018).
[23] Yu Feng, Ruben Martins, Jacob Van Geffen, Isil Dillig, and Swarat Chaudhuri. 2017. Component-based synthesis of table consolidation and transformation tasks from examples. *ACM SIGPLAN Notices* 52, 6 (2017), 422–436.
[24] Margarida Ferreira, Miguel Terra-Neves, Miguel Ventura, Inês Lynce, and Ruben Martins. 2021. FOREST: An interactive multi-tree synthesizer for regular expressions. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 152–169.
[25] Margarida Ferreira, Miguel Terra-Neves, Miguel Ventura, Inês Lynce, and Ruben Martins. 2021. FOREST: An Interactive Multi-tree Synthesizer for Regular Expressions. In *Tools and Algorithms for the Construction and Analysis of Systems*, Jan Friso Groote and Kim Guldstrand Larsen (Eds.). Springer International Publishing, Cham, 152–169.
[26] Aymeric Fromherz, Abdelraouf Ouadjaout, and Antoine Miné. 2018. Static value analysis of Python programs by abstract interpretation. In *NASA Formal Methods: 10th International Symposium, NFM 2018, Newport News, VA, USA, April 17-19, 2018, Proceedings 10*. Springer, 185–202.
[27] Alexander L Gaunt, Marc Brockschmidt, Nate Kushman, and Daniel Tarlow. 2017. Differentiable programs with neural libraries. In *International Conference on Machine Learning*. PMLR, 1213–1222.
[28] Isaac Gibbs and Emmanuel Candes. 2021. Adaptive conformal inference under distribution shift. *Advances in Neural Information Processing Systems* 34 (2021).
[29] Isaac Gibbs and Emmanuel Candès. 2021. Conformal inference for online prediction with arbitrary distribution shifts. In *NeurIPS*.
[30] Zheng Guo, Michael James, David Justo, Jiaxiao Zhou, Ziteng Wang, Ranjit Jhala, and Nadia Polikarpova. 2019. Program synthesis by type-guided abstraction refinement. *Proceedings of the ACM on Programming Languages* 4, POPL (2019), 1–28.
[31] Tanmay Gupta and Aniruddha Kembhavi. 2023. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14953–14962.
[32] Sankha Narayan Guria, Jeffrey S Foster, and David Van Horn. 2023. Absynthe: Abstract Interpretation-Guided Synthesis. *Proceedings of the ACM on Programming Languages* 7, PLDI (2023), 1584–1607.
[33] Shivam Handa and Martin Rinard. 2021. Program Synthesis Over Noisy Data with Guarantees. *arXiv preprint arXiv:2103.05030* (2021).
[34] Shivam Handa and Martin C Rinard. 2020. Inductive program synthesis over noisy data. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 87–98.
[35] Dan Hendrycks and Thomas Dietterich. 2019. Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261* (2019).
[36] Jiani Huang, Calvin Smith, Osbert Bastani, Rishabh Singh, Aws Albarghouthi, and Mayur Naik. 2020. Generating programmatic referring expressions via program synthesis. In *International Conference on Machine Learning*. PMLR, 4495–4506.
[37] Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. 2010. Oracle-guided component-based program synthesis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1* (Cape Town, South Africa) *(ICSE '10)*. Association for Computing Machinery, New York, NY, USA, 215–224. doi:10.1145/1806799.1806833
[38] Ruyi Ji, Chaozhe Kong, Yingfei Xiong, and Zhenjiang Hu. 2023. Improving Oracle-Guided Inductive Synthesis by Efficient Question Selection. *Proc. ACM Program. Lang.* 7, OOPSLA1, Article 103 (apr 2023), 29 pages. doi:10.1145/3586055
[39] Ruyi Ji, Jingjing Liang, Yingfei Xiong, Lu Zhang, and Zhenjiang Hu. 2020. Question selection for interactive program synthesis. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (London, UK) *(PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 1143–1158. doi:10.1145/3385412.3386025
[40] Chengyue Jiang, Zijian Jin, and Kewei Tu. 2021. Neuralizing regular expressions for slot filling. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 9481–9498.
[41] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. 2017. Inferring and executing programs for visual reasoning. In *Proceedings of the IEEE international conference on computer vision*. 2989–2998.
[42] Sven Keidel and Sebastian Erdweg. 2019. Sound and reusable components for abstract interpretation. *Proceedings of the ACM on Programming Languages* 3, OOPSLA (2019), 1–28.
[43] Alex Kendall, Matthew Grimes, and Roberto Cipolla. 2015. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*. 2938–2946.
[44] Larissa Laich, Pavol Bielik, and Martin Vechev. 2020. Guiding Program Synthesis by Learning to Generate Examples. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BJl07ySKvS

[45] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[46] Yen-Cheng Liu, Chih-Yao Ma, Zijian He, Chia-Wen Kuo, Kan Chen, Peizhao Zhang, Bichen Wu, Zsolt Kira, and Peter Vajda. 2021. Unbiased teacher for semi-supervised object detection. *arXiv preprint arXiv:2102.09480* (2021).

[47] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. 2018. Deepproblog: Neural probabilistic logic programming. *Advances in neural information processing systems* 31 (2018).

[48] Emanuele Marconato, Stefano Teso, Antonio Vergari, and Andrea Passerini. 2023. Not all neuro-symbolic concepts are created equal: Analysis and mitigation of reasoning shortcuts. *Advances in Neural Information Processing Systems* 36 (2023), 72507–72539.

[49] Benjamin Mariano, Yanju Chen, Yu Feng, Greg Durrett, and Işil Dillig. 2022. Automated transpilation of imperative to functional code using neural-guided program synthesis. *Proceedings of the ACM on Programming Languages* 6, OOPSLA1 (2022), 1–27.

[50] Mikaël Mayer, Gustavo Soares, Maxim Grechkin, Vu Le, Mark Marron, Oleksandr Polozov, Rishabh Singh, Benjamin Zorn, and Sumit Gulwani. 2015. User interaction models for disambiguation in programming by example. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. 291–301.

[51] Mikaël Mayer, Gustavo Soares, Maxim Grechkin, Vu Le, Mark Marron, Oleksandr Polozov, Rishabh Singh, Benjamin Zorn, and Sumit Gulwani. 2015. User Interaction Models for Disambiguation in Programming by Example. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (Charlotte, NC, USA) *(UIST '15)*. Association for Computing Machinery, New York, NY, USA, 291–301. doi:10.1145/2807442.2807459

[52] Stephen Mell, Steve Zdancewic, and Osbert Bastani. 2024. Optimal Program Synthesis via Abstract Interpretation. *Proceedings of the ACM on Programming Languages* 8, POPL (2024), 457–481.

[53] Manasij Mukherjee, Pranav Kant, Zhengyang Liu, and John Regehr. 2020. Dataflow-based pruning for speeding up superoptimization. *Proceedings of the ACM on Programming Languages* 4, OOPSLA (2020), 1–24.

[54] Aaditya Naik, Jason Liu, Claire Wang, Amish Sethi, Saikat Dutta, Mayur Naik, and Eric Wong. 2024. Dolphin: A programmable framework for scalable neurosymbolic learning. *arXiv preprint arXiv:2410.03348* (2024).

[55] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, Vol. 2011. Granada, 4.

[56] Liqiang Nie, Shuicheng Yan, Meng Wang, Richang Hong, and Tat-Seng Chua. 2012. Harvesting visual concepts for image search with complex queries. In *Proceedings of the 20th ACM International Conference on Multimedia* (Nara, Japan) *(MM '12)*. Association for Computing Machinery, New York, NY, USA, 59–68. doi:10.1145/2393347.2393363

[57] Saswat Padhi, Prateek Jain, Daniel Perelman, Oleksandr Polozov, Sumit Gulwani, and Todd Millstein. 2018. FlashProfile: a framework for synthesizing data profiles. *Proc. ACM Program. Lang.* 2, OOPSLA, Article 150 (Oct. 2018), 28 pages. doi:10.1145/3276520

[58] Shankara Pailoor, Yuepeng Wang, Xinyu Wang, and Isil Dillig. 2021. Synthesizing data structure refinements from integrity constraints. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 574–587.

[59] Sangdon Park, Edgar Dobriban, Insup Lee, and Osbert Bastani. 2021. PAC prediction sets under covariate shift. *arXiv preprint arXiv:2106.09848* (2021).

[60] Ramya Ramalingam, Sangdon Park, and Osbert Bastani. 2024. Uncertainty Quantification for Neurosymbolic Programs via Compositional Conformal Prediction. (2024). arXiv:2405.15912

[61] Veselin Raychev, Pavol Bielik, Martin Vechev, and Andreas Krause. 2016. Learning programs from noisy data. *ACM Sigplan Notices* 51, 1 (2016), 761–774.

[62] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B Gupta, Xiaojiang Chen, and Xin Wang. 2021. A survey of deep active learning. *ACM computing surveys (CSUR)* 54, 9 (2021), 1–40.

[63] Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. 2002. Parametric shape analysis via 3-valued logic. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 24, 3 (2002), 217–298.

[64] Greg Schohn and David Cohn. 2000. Less is more: Active learning with support vector machines. In *ICML*, Vol. 2. Citeseer, 6.

[65] Amazon Web Services. n.d.. *Amazon Rekognition Documentation.* https://docs.aws.amazon.com/rekognition/ Accessed: 2025-03-13.

[66] Burr Settles. 2009. Active learning literature survey. (2009).

[67] Glenn Shafer and Vladimir Vovk. 2008. A tutorial on conformal prediction. *Journal of Machine Learning Research* 9, Mar (2008), 371–421.

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

[69] Rishabh Singh and Armando Solar-Lezama. 2011. Synthesizing data structure manipulations from storyboards. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 289–299.

[70] Sunbeom So and Hakjoo Oh. 2017. Synthesizing imperative programs from examples guided by static analysis. In *International Static Analysis Symposium*. Springer, 364–381.

[71] Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. 2020. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *Advances in neural information processing systems* 33 (2020), 596–608.

[72] Chen Sun, Chuang Gan, and Ram Nevatia. 2015. Automatic Concept Discovery From Parallel Text and Visual Corpora. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

[73] Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 11888–11898.

[74] Ryan J Tibshirani, Rina Foygel Barber, Emmanuel Candes, and Aaditya Ramdas. 2019. Conformal prediction under covariate shift. *Advances in neural information processing systems* 32 (2019).

[75] Julien Valentin, Angela Dai, Matthias Nießner, Pushmeet Kohli, Philip Torr, Shahram Izadi, and Cem Keskin. 2016. Learning to navigate the energy landscape. In *2016 Fourth International Conference on 3D Vision (3DV)*. IEEE, 323–332.

[76] Lazar Valkov, Dipak Chaudhari, Akash Srivastava, Charles Sutton, and Swarat Chaudhuri. 2018. Houdini: Lifelong learning as program synthesis. *Advances in neural information processing systems* 31 (2018).

[77] Martin Vechev, Eran Yahav, and Greta Yorsh. 2010. Abstraction-guided synthesis of synchronization. In *Proceedings of the 37th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 327–338.

[78] Gust Verbruggen, Vu Le, and Sumit Gulwani. 2021. Semantic programming by example with pre-trained models. *Proceedings of the ACM on Programming Languages* 5, OOPSLA (2021), 1–25.

[79] Gust Verbruggen, Vu Le, and Sumit Gulwani. 2021. Semantic programming by example with pre-trained models. *Proceedings of the ACM on Programming Languages* 5, OOPSLA (2021), 1–25.

[80] Chenglong Wang, Alvin Cheung, and Rastislav Bodik. 2017. Interactive Query Synthesis from Input-Output Examples. In *Proceedings of the 2017 ACM International Conference on Management of Data* (Chicago, Illinois, USA) *(SIGMOD '17)*. Association for Computing Machinery, New York, NY, USA, 1631–1634. doi:10.1145/3035918.3058738

[81] Chenglong Wang, Alvin Cheung, and Rastislav Bodik. 2017. Synthesizing highly expressive SQL queries from input-output examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 452–466.

[82] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

[83] Xinyu Wang, Greg Anderson, Isil Dillig, and Kenneth L McMillan. 2018. Learning abstractions for program synthesis. In *Computer Aided Verification: 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I 30*. Springer, 407–426.

[84] Xinyu Wang, Isil Dillig, and Rishabh Singh. 2017. Program synthesis using abstraction refinement. *Proceedings of the ACM on Programming Languages* 2, POPL (2017), 1–30.

[85] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155* (2023).

[86] Chen Xu and Yao Xie. 2021. Conformal prediction interval for dynamic time-series. In *Proceedings of the 38th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*. PMLR, 11559–11569.

[87] Yongho Yoon, Woosuk Lee, and Kwangkeun Yi. 2023. Inductive program synthesis via iterative forward-backward abstract interpretation. *Proceedings of the ACM on Programming Languages* 7, PLDI (2023), 1657–1681.

A Proofs

Lemma A.1. Let P_1, P_2 be a pair of programs such that $[\![P_1]\!]_\Phi(I) \neq [\![P_2]\!]_\Phi(I)$ for some $I \in \mathcal{I}$, where $\mathcal{I}$ denotes the input space. Then, $P_1 \not\simeq_\Phi P_2$.

Proof. Note that Possible($\varnothing$) contains a single formula, true. From the assumption, this implies

$$[\![P_1]\!]_{\Phi \wedge \mathsf{true}}(I) \neq [\![P_2]\!]_{\Phi \wedge \mathsf{true}}(I) \tag{5}$$

Hence, $P_1 \not\simeq_\Phi P_2$ □

Lemma A.2. Let $\mathcal{P}$ be a hypothesis space and let P' be a program randomly sampled from P. For any user feedback Φ, input space $\mathcal{I}$ and question space $\mathcal{Q}$, then we have:

$$\forall P_1, P_2 \in \mathcal{P}.\ P_1 \simeq_\Phi P_2$$

if and only if Distinguish($P', \mathcal{P} \backslash P', \Phi, \mathcal{I}, \mathcal{Q}$) returns false.

Proof. $\Rightarrow$ Suppose $\forall P_1, P_2 \in \mathcal{P}.\ P_1 \simeq_\Phi P_2$. Then for any input I and any set of questions $\mathcal{Q}' \subseteq \mathcal{Q}$,

$$\forall \Phi' \in \mathsf{Possible}(\mathcal{Q}').\ [\![P_1]\!]_{\Phi \wedge \Phi'}(I) = [\![P_2]\!]_{\Phi \wedge \Phi'}(I), \tag{6}$$

and by extension,

$$\forall \Phi' \in \mathsf{Possible}(\mathcal{Q}').\ \mathrm{CCE}(P, I, \Phi \wedge \Phi') = \mathrm{CCE}(P_2, I, \Phi \wedge \Phi'). \tag{7}$$

Therefore, Distinguish($P', \mathcal{P}, \Phi, I, \mathcal{Q}$) will return false for any $P' \sim \mathcal{P}$.

$\Leftarrow$ Suppose Distinguish($P', \mathcal{P} \backslash P', \Phi, \mathcal{I}, \mathcal{Q}$) returns false. Towards contradiction, suppose that there exists $P_1, P_2 \in \mathcal{P}$ such that $P_1 \not\simeq_\Phi P_2$. Then there exists $\Phi' \in \mathsf{Possible}(\mathcal{Q})$ and some input I such that $[\![P_1]\!]_{\Phi \wedge \Phi'}(I) = [\![P_2]\!]_{\Phi \wedge \Phi'}(I)$, and by extension $\mathrm{CCE}(P_1, I, \Phi \wedge \Phi') = \mathrm{CCE}(P_2, I, \Phi \wedge \Phi')$. Since Distinguish enumerates all possible sets of questions and all possible answers, the procedure must at some point check that $\mathrm{CCE}(P_1, I, \Phi \wedge \Phi') = \mathrm{CCE}(P', I, \Phi \wedge \Phi')$ and $\mathrm{CCE}(P_2, I, \Phi \wedge \Phi') = \mathrm{CCE}(P', I, \Phi \wedge \Phi')$. However, this implies $\mathrm{CCE}(P_1, I, \Phi \wedge \Phi') = \mathrm{CCE}(P_2, I, \Phi \wedge \Phi')$, which is a contradiction. Therefore, $\forall P_1, P_2 \in \mathcal{P}.\ P_1 \simeq_\Phi P_2$. □

Lemma A.3. Let P_1, P_2 be a pair of programs such that $P_1 \simeq_\Phi P_2$. Then, assuming Φ represents accurate user feedback, we have $\forall I \in \mathcal{I}.\ (\!|P_1|\!)(I) = (\!|P_2|\!)(I)$ where $\mathcal{I}$ denotes the input space.

Proof. Towards contradiction, suppose there exist programs P_1, P_2 and an input I^* such that

$$(\!|P_1|\!)(I^*) \neq (\!|P_2|\!)(I^*).$$

Consider the set of labels $\mathcal{F} = \{(f, I, (\!|f|\!)(I)) \mid f \in \mathcal{N}, I \in \mathcal{I}\}$. This set provides the ground truth label for every neural function on every input. Constrained conformal semantics with the formula $\Phi \wedge \Phi_\mathcal{F}$ is thus equivalent to the ground truth semantics. Then, since $P_1 \simeq P_2$, their output under constrained conformal semantics with feedback $\Phi \wedge \Phi_\mathcal{F}$ must be equal on all inputs. That is,

$$(\!|P_1|\!)(I^*) = [\![P_1]\!]_{\Phi \wedge \Phi_\mathcal{F}}(I^*) = [\![P_2]\!]_{\Phi \wedge \Phi_\mathcal{F}}(I^*) = (\!|P_2|\!)(I^*),$$

which contradicts the initial assumption.

□

Theorem A.1. Let P^* be the ground truth program, and let P be the program returned by the ActiveLearning procedure. Then, under the assumption that $P^* \in \mathcal{P}$:

$$\forall I \in \mathcal{I}.\ (\!|P|\!)(I) = (\!|P^*|\!)(I)$$

Proof. Note that P is a program selected from a refined hypothesis space $\mathcal{P}' \subseteq \mathcal{P}$ such that, for some user feedback Φ and some $P' \in \mathcal{P}$, Distinguish($P', \mathcal{P} \backslash P', \Phi, \mathcal{I}, \mathcal{Q}$) returns false. By Lemma 4.1, $P_1 \simeq_\Phi P_2$ for all pairs $(P_1, P_2) \in \mathcal{P}' \times \mathcal{P}'$. Since the user always gives feedback consistent with the ground truth, $P^* \in \mathcal{P}'$. Then $P \simeq_\Phi P^*$. Therefore, by Lemma 3.1, $(\!|P|\!)(I) = (\!|P^*|\!)(I)$ for all $I \in \mathcal{I}$. □

Theorem A.2. Let $P = (\text{def } f_{\text{synth}}(x) = S)$ be a program and let Φ denote user feedback. Then, $P \Vdash_k \Phi$ implies $P \models \Phi$.

Proof. To prove $P \models \Phi_{\mathcal{F}}$, we must show $\forall (f_{\text{synth}}, I, O) \in \mathcal{F}.\ O \in [\![P]\!]_{\Phi_{\mathcal{F}}}(I)$. Since $P \Vdash_k \Phi_{\mathcal{F}}$, we know $\forall (f_{\text{synth}}, I, O) \in \mathcal{F}.\ O \in [\![P]\!]^k_{\Phi_{\mathcal{F}}}(I)$. Since BCE simply samples subsets of the prediction sets output by CCE, it must be that $[\![P]\!]^k_{\Phi_{\mathcal{F}}}(I) \subseteq [\![P]\!]_{\Phi_{\mathcal{F}}}(I)$, and so $O \in [\![P]\!]_{\Phi_{\mathcal{F}}}(I)$. Therefore, $P \models \Phi_{\mathcal{F}}$ □

Corollary A.3. *Let $\mathcal{P}$ be a set of programs and let Φ denote user feedback. Then, we have:*

$$(\mathcal{P} \downarrow \Phi) \ \supseteq\ \{P \in \mathcal{P} \mid P \Vdash \Phi\}$$

Proof. Let $P' \in \{P \in \mathcal{P} \mid P \Vdash \Phi\}$. Since $P' \Vdash \Phi$, by Theorem A.2, $P' \models \Phi$. Then $P' \in \mathcal{P} \downarrow \Phi$, and hence $(\mathcal{P} \downarrow \Phi) \ \supseteq\ \{P \in \mathcal{P} \mid P \Vdash \Phi$. □

Corollary A.4. *Let $q = (f, I)$ be a question and $\mathcal{P}$ be a program space consistent with Φ. Then:*

$$\Pi(q, \mathcal{P}) \leq \frac{|\mathcal{P}| - \max\limits_i |\{P \in \mathcal{P} \mid P \Vdash \Phi \wedge f(I) = a_i\}|}{|\mathcal{P}|}$$

Proof. By Corollary A.3, $\forall I \in \mathcal{I}.\ |\{P \in \mathcal{P} \mid P \Vdash \Phi \wedge f(I) = a_i\}| \leq |(\mathcal{P} \downarrow \Phi \wedge f(I) = a_i)|$. Then $\max_i |\{P \in \mathcal{P} \mid P \Vdash \Phi \wedge f(I) = a_i\}| \leq \max_i |(\mathcal{P} \downarrow \Phi \wedge f(I) = a_i)|$, and hence

$$\frac{|\mathcal{P}| - \max\limits_i |(\mathcal{P} \downarrow \Phi \wedge f(I) = a_i)|}{|\mathcal{P}|} \leq \frac{|\mathcal{P}| - \max\limits_i |\{P \in \mathcal{P} \mid P \Vdash \Phi \wedge f(I) = a_i\}|}{|\mathcal{P}|}.$$

□

Theorem A.5. The SelectQuestion procedure returns a question $q^* \in \mathcal{Q}$ with the highest pruning power modulo hypothesis space $\mathcal{P}$. That is,

$$\forall q \in \mathcal{Q}.\ \Pi(q, \mathcal{P}) \leq \Pi(q^*, \mathcal{P})$$

Proof. Note that SelectQuestion returns a question $q^* \in \mathcal{Q}$ such that, for all questions $q \in \mathcal{Q}$, either we have computed the pruning power of q and checked that $\Pi(q, \mathcal{P}) \leq \Pi(q^*, \mathcal{P})$, or we have checked that the pruning power of q^* is greater than the approximated pruning power of q:

$$\frac{|\mathcal{P}| - \max\limits_i |\{P \in \mathcal{P} \mid P \Vdash \Phi \wedge f(I) = a_i\}|}{|\mathcal{P}|} \leq \Pi(q^*, \mathcal{P}).$$

By Corollary A.4,

$$\Pi(q, \mathcal{P}) \leq \frac{|\mathcal{P}| - \max\limits_i |\{P \in \mathcal{P} \mid P \Vdash \Phi \wedge f(I) = a_i\}|}{|\mathcal{P}|}.$$

Therefore,

$$\forall q \in \mathcal{Q}.\ \Pi(q, \mathcal{P}) \leq \Pi(q^*, \mathcal{P}).$$

□

$$
\begin{array}{rl}
P := & \lambda x.\ \text{let } y = \textbf{Segment}(x) \text{ in } E \\
E := & y \mid \text{Filter}(E, f) \mid \text{Complement}(E) \\
& \mid \text{Union}_N(E_1, \cdots, E_N) \\
& \mid \text{Intersect}_N(E_1, \cdots, E_N) \\
& \mid E \times E \\
f := & \textbf{HasAttribute}(\texttt{a}) \mid \text{HasRelation}(\texttt{r})
\end{array}
$$

Fig. 22. The ObjExtract DSL. The neural components are in bold.

Fig. 23. Image segments produced via object detection.

B Hyperparameter Selection

Recall that the bounded conformal evaluation strategy used in question selection samples subsets of the prediction sets output by a program and its intermediary operations. In the VisArith domain, the BCE hyper-parameter k was set to 1, and we found that increasing it beyond $k = 1$ is not useful. In the ImageEdit and ImageSearch domains, on the other hand, we found that proportionally sampling 5% of the elements in the prediction set gives the best results until a k limit is reached. In particular, if the prediction set contains n elements, we choose $max(n/20, k)$ elements, and we use a k value of 10 in our experiments.

C Details About Application Domains

In Section 6, we evaluate SmartLabel on three application domains: ImageEdit, ImageSearch, and VisArith. In this section, we provide more details about these domains and their abstractions.

C.1 Intermediate Representation for Image Editing and Search

While prior work proposes different DSLs targeting image editing versus image search, these domains both use the same neural networks for image segmentation and classification. Hence, to avoid defining two different abstract domains, we translate both DSLs proposed in prior work to an intermediate representation (IR) shown in Figure 22 (henceforth referred to as ObjExtract) used for retrieving objects from an image. The ImageEdit DSL extracts one or more sets of objects using the DSL in Figure 22 and applies the specified operation to each set. Meanwhile, the ImageSearch DSL returns a boolean depending on whether or not the IR in Figure 22 returns the empty set.

A program P in this DSL is an object extraction function of the form $\lambda x.$ let $y = \textbf{Segment}(x)$ in E, where **Segment** is a neural component for performing *segmentation* on the input image x. In particular, the **Segment** operation outputs a set of *image segments* $o = (\phi, \Delta)$, where ϕ denotes an image corresponding to a specific object and Δ is the location of that object within the original image. Following prior work [7], we represent the location of the object as a bounding box (j_l, j_r, j_t, j_b) describing the left, right, top, and bottom pixels of the object. For instance, Figure 23 shows the result of running segmentation on an example image.

As shown in Figure 22, expressions E in the ObjExtract DSL involve a combination of neural and symbolic operations. Specifically, symbolic expressions include standard set operations (complement, union, intersection, and Cartesian product) with well-known semantics. The Filter construct takes as input a set of image segments and a predicate f, which can be either neural or symbolic. In particular, **HasAttribute**(a) is a neural component that checks whether a given image segment has a specific attribute a, such as `face` or `guitar`, indicating that this object is labeled as a face or guitar. On the other hand, HasRelation(r) is a symbolic binary predicate for checking whether a pair of image segments have a certain spatial relation (e.g., `below`). Since spatial relations can be determined based on bounding boxes, evaluation of such predicates does not require the use of a neural network.

Example C.1. Consider the following expression in the ObjExtract DSL:

$$\mathsf{Intersection}(\mathsf{Filter}(y, \textbf{HasAttribute}((\texttt{face}))), \mathsf{Complement}(\mathsf{Filter}(y, \textbf{HasAttribute}((\texttt{smiling})))))$$

This expression yields all image segments containing human faces that are not smiling. For instance, applying this expression to the segments from Figure 23 would yield object ϕ_1.

We conclude this section by showing how to express a few representative tasks from the image editing and image search domains using our IR:

Example C.2. Consider the following program in the ImageEdit DSL:

$$\{\mathsf{Intersect}(\textbf{Is}(\textbf{Object}(\texttt{face})), \mathsf{Complement}(\mathsf{Find}((\textbf{Is}(\textbf{Object}(\texttt{guitar})), \textbf{Is}(\textbf{Object}(\texttt{face})), \mathsf{GetAbove})))) \rightarrow \mathsf{Blur}\}.$$

This program blurs the faces of all people who are not holding guitars. We may instead express this task by applying a blur action to all objects extracted by the following ObjExtract expression:

$$\mathsf{Intersect}(\mathsf{Filter}(y, \textbf{HasAttribute}(\texttt{face})), \mathsf{Complement}(\mathsf{Filter}(\mathsf{Filter}(y, \textbf{HasAttribute}(\texttt{guitar})) \times \mathsf{Filter}(y, \textbf{HasAttribute}(\texttt{face})), \mathsf{HasRelation}(\mathsf{above})))).$$

Example C.3. Consider the following program in the ImageSearch DSL:

$$\exists x.\exists y.\exists z.\textbf{HasType}(x, \texttt{bicycle}) \wedge \textbf{HasType}(y, \texttt{person}) \wedge \textbf{HasType}(z, \texttt{helmet}) \wedge \mathsf{HasRelation}(x, y, \texttt{below}) \wedge \mathsf{HasRelation}(y, z, \texttt{below}).$$

This program returns all images in a corpus that contain cyclists wearing helmets. We may instead express this task by applying the following ObjExtract expression to all images in the corpus, and returning those whose extracted object sets are non-empty:

$$\mathsf{Filter}(\mathsf{Filter}(\mathsf{Filter}(y, \textbf{HasAttribute}(\texttt{bicycle})) \times \mathsf{Filter}(y, \textbf{HasAttribute}(\texttt{person})), \mathsf{HasRelation}(\mathsf{above})) \times \mathsf{Filter}(y, \textbf{HasAttribute}(\texttt{helmet})), \mathsf{HasRelation}(\mathsf{above})).$$

C.2 Abstract Domain for ObjExtract Intermediate Language

Since programs in the ObjExtract IR return a *set* of objects under the regular evaluation semantics, they must return *sets of sets* under the conformal semantics. We thus consider a *set interval* abstract domain where each set $\tilde{O}$ of object sets is abstracted using a *set interval* $[O^-, O^+]$ satisfying $\forall O \in \tilde{O}.\ O^- \subseteq O \subseteq O^+$. The abstraction and concretization functions for the set interval domain are defined as follows:

$$\alpha(\tilde{O}) = [\{o \mid \forall O \in \tilde{O}.\ o \in O\}, \{o \mid \exists O \in \tilde{O}.\ o \in O\}]$$
$$\gamma([O^-, O^+]) = \{O \mid O^- \subseteq O \subseteq O^+\}$$

In other words, the abstraction function α takes a set $\tilde{O}$ of object sets and outputs an interval $[O^-, O^+]$ where O^- (resp. O^+) is an under-approximation (resp. over-approximation of every object set $O \in \tilde{O}$. Conversely, the concretization function takes an abstract value $[O^-, O^+]$ and outputs the set of all object sets O that are a superset of O^- and a subset of O^+.

$$
\begin{aligned}
[\![\textbf{Segment}]\!]^\sharp(I) &= \alpha([\![\textbf{Segment}]\!]_\Phi(I)) \\
[\![\textbf{HasAttribute}(\mathsf{a})]\!]^\sharp(I) &= \alpha([\![\textbf{HasAttribute}(\mathsf{a})]\!]_\Phi(I)) \\
[\![\mathsf{Union}(E_1,\ldots,E_N)]\!]^\sharp(I) &= \left[\bigcup_i O_i^-, \bigcup_i O_i^+\right], \text{where } [O_i^-, O_i^+] = [\![E_i]\!]^\sharp(I) \\
[\![\mathsf{Intersect}(E_1,\ldots,E_N)]\!]^\sharp(I) &= \left[\bigcap_i O_i^-, \bigcap_i O_i^+\right], \text{where } [O_i^-, O_i^+] = [\![E_i]\!]^\sharp(I) \\
[\![\mathsf{Complement}(E)]\!]^\sharp(I) &= [U^- \setminus O^+, U^+ \setminus O^-] \\
&\quad \text{where } [O^-, O^+] = [\![E]\!]^\sharp(I) \text{ and } [U^-, U^+] = [\![\textbf{Segment}]\!]^\sharp(I) \\
[\![\mathsf{Filter}(E, \textbf{HasAttribute}(\mathsf{a}))]\!]^\sharp(I) &= [\{o \mid o \in O_1^- \wedge o \in O_2^-\}, \{o \mid o \in O_1^+ \wedge o \in O_2^+\}] \\
&\quad \text{where } [O_1^-, O_1^+] = [\![E]\!]^\sharp(I) \text{ and } [O_2^-, O_2^+] = [\![\textbf{HasAttribute}(\mathsf{a})]\!]^\sharp(I) \\
[\![\mathsf{Filter}(E_1 \times E_2, \mathsf{HasRelation}(\mathsf{r}))]\!]^\sharp(I) &= [\{(o_1,o_2) \in O_1^- \times O_2^- \mid [\![\mathsf{HasRelation}(\mathsf{r})]\!](o_1,o_2)\}, \\
&\qquad \{(o_1,o_2) \in O_1^+ \times O_2^+ \mid [\![\mathsf{HasRelation}(\mathsf{r})]\!](o_1,o_2)\}] \\
&\quad \text{where } [O_i^-, O_i^+] = [\![E_i]\!]^\sharp(I)
\end{aligned}
$$

Fig. 24. Abstract forward semantics for ObjExtract.

Forward abstract semantics. Figure 24 presents the forward abstract semantics for this domain. The first two rules for neural components apply the abstraction function to the conformal prediction result. The transformer for Union first computes the abstract values for the arguments and then takes the union of the lower and upper bounds. The transformer for Intersect is similar to that of Union, but it takes the intersections of the lower and upper bounds instead of their unions. The rule for Complement first computes the abstract value $[O^-, O^+]$ for its argument E, and then computes the new lower and upper bounds as $U^- \setminus O^+$ and $U^+ \setminus O^-$. In the rule for Filter, the under- (resp. over-) approximation for the whole expression includes an object o if (1) it is in the under- (resp. over-) approximation of the nested expression E and (2) if o must (resp. may) have attribute a according to conformal prediction. The final rule shows the abstract transformer for filtering based on spatial relations between a pair of objects. This rule is similar to the previous one, but uses a symbolic predicate instead of a neural one.

Backward abstract semantics. Given an operator f whose abstract output is α_o and whose abstract inputs are $\alpha_1, \ldots, \alpha_k$, the inverse abstract transformer $[\![f]\!]^{\sharp^{-i}}$ infers a new abstract value α_i' for the i'th child of f such that:

$$
\forall \overline{x}, y. \left(\left(f(x_1,\ldots,x_n) = y \wedge y \in \gamma(\alpha_o) \wedge \bigwedge_{j=1}^{n} x_j \in \gamma(\alpha_j) \right) \implies x_i \in \gamma([\![f]\!]^{\sharp^{-i}}(\alpha_o, \alpha_1, \ldots, \alpha_n)) \right)
$$

The idea is to use the abstract backward semantics to tighten the abstract value of the i'th argument of f as $\alpha_i \sqcap [\![f]\!]^{\sharp^{-i}}(\alpha_o, \alpha_1, \ldots, \alpha_n)$. Figure 25 presents the backward abstract transformers for ObjExtract. To understand the rule for Union, suppose we have $A = A_1 \cup A_2$. Since $A \supseteq A_1 \supseteq A \setminus A_2$, we can derive $A \setminus A_2$ as a lower bound for A_1 and A as an upper bound. Generalizing this to more than two operands, the union rule computes the upper bound for the i'th operand of union as O^+ and the lower bound as $O^- \setminus \cup_{i \neq j} O_j^+$.

To understand the rule for Filter, suppose $A = \mathsf{Filter}(A_1, A_2)$. Then $A \subseteq A_1 \subseteq A \cup A_2$. Hence, we can use A as a lower bound on A_1 and $A \cup A_2$ as an upper bound. Thus, the Filter rule computes the lower bound for the i'th operand as O^- and the upper bound as $O^+ \cup (U^+ \setminus O_j^-)$ (with $j \neq i$). The

$$[\![\mathsf{Union}(\ldots)]\!]^{\sharp -i}([O^-, O^+], [O_1^-, O_1^+], \ldots, [O_n^-, O_n^+]) = \left[O^- \setminus \bigcup_{j \neq i} O_j^+,\ O^+\right]$$

$$[\![\mathsf{Intersect}(\ldots)]\!]^{\sharp -i}([O^-, O^+], [O_1^-, O_1^+], \ldots, [O_n^-, O_n^+]) = \left[O^-,\ O^+ \cup \left(U^+ \setminus (\bigcap_{j \neq i} O_j^-)\right)\right]$$

$$[\![\mathsf{Complement}(E)]\!]^{\sharp -1}([O^-, O^+], \ldots) = [U^- \setminus O^+, U^+ \setminus O^-]$$

$$[\![\mathsf{Filter}(E, \textbf{HasAttribute}(\mathsf{a}))]\!]^{\sharp -i}([O^-, O^+], [O_1^-, O_1^+], [O_2^-, O_2^+]) = [O^-, O^+ \cup (U^+ \backslash O_j^-)] \text{ where } j \neq i$$

$$[\![\mathsf{Filter}(E, \mathsf{HasRelation}(\mathsf{r}))]\!]^{\sharp -1}([O^-, O^+]) = [O^-, U^+ \times U^+]$$

$$[\![E_1 \times E_2]\!]^{\sharp -i}([O^-, O^+]) = [O^- \downarrow i, O^+ \downarrow i]$$

Fig. 25. Abstract backward semantics for ObjExtract where $[U^-, U^+] = [\![\textbf{Segment}]\!]^\sharp(I)$. We use the notation $O \downarrow j$ to denote the set $\{o_j \mid (o_1, \ldots, o_n) \in O\}$.

second filter rule for binary predicates is similar but yields the upper bound $U^+ \times U^+$, since such an operation can only be applied to pairs of objects. The rule for Intersect is a generalization of the rule for Filter. The rule for Complement is the same as the abstract transformer in the forward direction, owing to the face that the set complement operation is involutive.

C.3 Further Details About the ImageEdit and ImageSearch Domains

In this subsection, we provide further details about some implementation choices for the ImageEdit and ImageSearch domains.

Calibration dataset. As discussed in Section 5, conformal prediction requires calibration datasets for which ground truth labels are available and that have a similar data distribution as the input space. Unlike the VisArith domain, the ImageEdit and ImageSearch domains lacks quality datasets for which ground truth labels for the objects and attributes are available. To deal with the lack of a labeled calibration set, we use the following methodology. For a given image I, we use the predictions of the underlying neural networks for segmentation and classification as the ground truth labels. We then distort I using existing image perturbation techniques [35] to produce a new image I' and generate predicted labels for I by using the classification results on I'. We use the same strategy to obtain pseudo-ground truth labels for both the calibration and test set. Using pseudo-labels for both training and testing data is an established methodology in Computer Vision [43, 46, 71, 75].

Object detection and classification. The ImageEdit and ImageSearch DSLs contain operators that detect and label object in an image (e.g. the **Is** operator in the ImageEdit DSL and the **HasAttribute** operator in the ImageSearch DSL). As in [7], we implement these operators using Amazon Rekognition. When a new batch of images is loaded, we perform a preprocessing step that computes the prediction sets of each neural attribute of each image, then memoizes these results to be used during active learning. Preprocessing takes an average of 1.6s per image.

C.4 Abstract Domain for Visual Arithmetic Application

For visual arithmetic tasks, we consider the DSL shown in Figure 19. Programs in this DSL take as input a list of images of handwritten digits, transform those images into integers using the neural function **`toDigit`**, and then perform list operations using the higher-order functions `fold`, `map`, and `filter`. A `fold` operation can use the binary functions `sum`, `max`, `product`, and `inc`. A `map` operation can use curried versions of any of these functions, as well as the neural function **`toDigit`**

$$
\begin{aligned}
[\![\texttt{toDigit}]\!]^\sharp(x) &= \alpha([\![\texttt{toDigit}]\!]_\Phi(x)) \\
[\![f]\!]^\sharp(([a,b],z),([a',b'],z')) &= ([[\![f]\!](a,a'),[\![f]\!](b,b')],z \wedge z') \\
[\![\lambda x.x < c]\!]^\sharp(([a,b],z)) &= \begin{cases} ([a,b],z) & \text{if } b < c \\ ([a,b],\text{false}) & \text{if } a \geq c \\ ([a,b],z \wedge *) & \text{otherwise} \end{cases} \\
[\![\texttt{filter}\ h\ E]\!]^\sharp &= \text{filter}\ [\![h]\!]^\sharp\ [\![E]\!]^\sharp \\
[\![\texttt{map}\ g\ E]\!]^\sharp &= \text{map}\ [\![g]\!]^\sharp\ [\![E]\!]^\sharp \\
[\![\texttt{fold}\ f\ c\ E]\!]^\sharp &= ([a_1,b_2],\text{true}) \\
&\quad \text{where}\ \ ([a_1,b_1],z_1) = \text{fold}\ [\![f]\!]^\sharp\ \alpha(c)\ (\text{filter}\ (\lambda(x,y).y = \text{true})\ [\![E]\!]^\sharp) \\
&\quad \text{and}\ \ ([a_2,b_2],z_2) = \text{fold}\ [\![f]\!]^\sharp\ \alpha(c)\ (\text{filter}\ (\lambda(x,y).\ y \neq \text{false})\ [\![E]\!]^\sharp)
\end{aligned}
$$

Fig. 26. Abstract forward semantics for the VisArith DSL.

that predicts the label of an image. A `filter` operation takes in a boolean predicate of the form $\lambda x.x \lhd c$ where $\lhd \in \{\leq, =, \geq, \ldots\}$.

Abstract domain. Our abstraction represents a list of integers as a list of tuples (α_I^i, z^i) where α_I^i is the interval abstraction of an integer and $z^i \in \{\text{true}, \text{false}, *\}$ is a three-valued logic element that denotes whether or not the corresponding element is *definitely* in the list. Intuitively, if z^i is $*$, then this element may or may not have been filtered from the list. If z^i is false, then the element has definitely been removed, and if z^i is true then it is definitely still in the list.

Forward abstract semantics. Figure 26 presents the abstract semantics for this DSL. In the following discussion, we explain each of the forward abstract transformers in more detail.

Binary operation. A binary operation f (e.g. addition) may be performed on two abstract values $([a,b],z)$, $([a',b'],z')$ by performing the operation on both the lower and upper bounds of the interval and taking the logical conjunction of z and z' under the 3-valued logic semantics [63].

Filter and map. A filtering function h may be performed on an abstract value $([a,b],z)$ by considering where *any*, *all*, or *no* concrete values in $\gamma(([a,b],z))$ would be filtered by h. For instance, consider the filtering function $\lambda x.x < c$. In the case that $a \geq c$, we know that no concrete values in $\gamma(([a,b],z))$ are filtered, and so h outputs the unchanged input $([a,b],z)$. In the case that $b < c$, we know that every concrete value in $\gamma(([a,b],z))$ are filtered, and so h outputs $([a,b],\text{false})$. In any other case, we do not know whether *some* concrete value is filtered, so we output $([a,b],*)$ to account for this uncertainty. To evaluate `filter` $h\ E$ abstractly, we simply filter the abstract list $[\![E]\!]^\sharp$ using $[\![h]\!]^\sharp$. The evaluation of `map` $g\ E$ is similar to the evaluation of `filter` $h\ E$.

Fold operation. Evaluating `fold` $c\ E$ abstractly is slightly more complex, as it requires computing an interval $[a',b']$ representing the minimum and maximum output of the `fold` operation. Our abstract semantics relies on the fact that all list elements are natural numbers and that, for all functions f in our DSL, we have $\texttt{fold}\ f\ c\ x : xs \geq \texttt{fold}\ f\ c\ xs$. Thus, we can compute the upper bound of the result as:

$$[\![f]\!]^\sharp\ \alpha(c)\ \text{filter}\ (\lambda(x,y).y \neq \text{false})\ [\![E]\!]^\sharp)$$

$$[\![\texttt{toDigit}]\!]^{\sharp^{-1}}(([a,b],z)) = ([a,b],z)$$

$$[\![\lambda x.x < c]\!]^{\sharp^{-1}}(([a,b],z)) = \begin{cases} ([a,c-1],z) & \text{if } z = \text{true} \\ ([a,b],z) & \text{otherwise} \end{cases}$$

$$[\![\texttt{filter}\ h\ E]\!]^{\sharp^{-2}}(l,\ldots) = \text{map}\ [\![h]\!]^{\sharp^{-1}}\ l$$

$$[\![\texttt{map}\ g\ E]\!]^{\sharp^{-2}}(l,\ldots) = \text{map}\ [\![g]\!]^{\sharp^{-1}}\ l$$

$$[\![\texttt{fold sum}\ c\ E\ ]\!]^{\sharp^{-3}}([a,b],[([a_1,b_1],z_1),\ldots,([a_n,b_n],z_n)]) = [([a'_1,b'_1],z_1),\ldots,([a'_n,b'_n],z_n)]$$
$$\text{where } [a'_i,b'_i] = [\max(0, a - c - \sum_{j\neq i} b_j), b - c - \sum_{j\neq i} a_j \cdot \mathbb{1}(z_j)]$$

Fig. 27. Abstract backward semantics for the VisArith DSL.

However, to compute a true lower bound, we should only consider the list elements that *must* be in the input list; hence, we compute the lower bound as

$$[\![f]\!]^{\sharp}\ \alpha(c)\ \text{filter}(\lambda(x,y).y = \text{true})[\![E]\!]^{\sharp}$$

Remark. Note that our forward abstract semantics preserve the length of an input list. In particular, if some operation could end up removing an element from the list, the forward semantics retains that element in the list but sets it corresponding boolean to either false or $*$. Our backward semantics rely on this length-preservation property of the abstract domain.

Backward abstract semantics. The backward abstract semantics for the VisArith domain are given in Figure 27. We explain each of the non-trivial rules in more detail below.

Predicates. Consider a predicate $\lambda x.x < c$ where x is a list element. If x is known to be *definitely* in the list after applying this predicate to x, then c constitutes a lower bound on x, as it was definitely *not* removed from the list by applying this predicate. Thus, as shown in Figure 27, we perform a case split on z, tightening the bound to $[c,b]$ when z is true and leaving it as $[a,b]$ otherwise.

Filter and map. The definition of $[\![\texttt{filter}]\!]^{\sharp^{-2}}$ relies on the backward semantics for predicates. For each element x in l, the backward semantics computes the input value of x as $[\![h]\!]^{\sharp^{-1}}(x)$. Note that the correctness of the backward semantics relies on the fact that our list abstract domain is length-preserving, as mentioned earlier. The definition of $[\![\texttt{map}]\!]^{\sharp^{-2}}$ is similar and relies on the abstract backward semantics $[\![g]\!]^{\sharp^{-1}}$. For example, $[\![\texttt{curry sum}\ c]\!]^{\sharp^{-1}}$ is defined as $\lambda x.x - c$. Thus, $[\![\texttt{map curry sum c L}]\!]^{\sharp^{-2}}$ ends up subtracting c from each element of the output list to compute the abstract value for input list L.

Fold. The final rule in Figure 27 defines the backward abstract semantics of `fold`. However, since precisely reasoning about `fold` requires performing a case split on the function argument f, we only show the case where $f = \texttt{sum}$ as a representative example. This rule shows how to compute the updated abstract value for the input list given that the output of the `fold` operation is $[a,b]$ and the initial abstract value of the input list is $[([a_1,b_1],z_1),\ldots([a_n,b_n],z_n)$. To understand this rule, suppose that the actual input list of fold has elements $[x_1,\ldots,x_n]$ and suppose that the actual output is y. Then, we have

$$x_i = y - c - \sum_{j\neq i} x_j$$

Thus, the lower bound a'_i and upper bound b'_i on x_i can be computed as:

$$a'_i = a - c - \sum_{j \neq i} b_j \quad b'_i = b - c - \sum_{j \neq i} a_j \times \mathbb{1}(z_i)$$

Note that, for the upper bound computation, we define $\mathbb{1}(z_i)$ to be 1 if z_i is true and 0 otherwise, and we only subtract a_i if $\mathbb{1}(z_i)$ is 1. This is needed to ensure that the value we compute is a true upper bound.

D List of ImageEdit Benchmarks

(1) {Intersect(Is(Smiling), Is(EyesOpen)) →Brighten}
Description: Brighten all faces that are smiling and have eyes open.
Dataset: Wedding

(2) {Find(Is(Object(face), Is(Object(face)), GetAbove) → Brighten}
Description: Brighten all faces in back.
Dataset: Wedding

(3) {Union(Is(Object(bride)), Is(Object(groom)) → Crop}
Description: Crop image to feature just faces of bride and groom.
Dataset: Wedding

(4) {Intersect(Is(Object(face)),
Complement(Is(Object(bride)) → Blur}
Description: Blur all faces except the bride's face.
Dataset: Wedding

(5) {Find(Find(Is(Object(face)), Is(Object(face)), GetRight), Is(Object(face))), GetRight) → Brighten}
Description: Brighten all faces except the leftmost two faces.
Dataset: Wedding

(6) {Intersect(Is(Object (face)), Complement(Intersect(Is(Smiling), Is(EyesOpen)))) → Blur }
Description: Blur all faces that are not smiling and do not have their eyes open.
Dataset: Wedding

(7) {Intersect(Is(Smiling), Is(EyesOpen), Complement(Is(Object(groom))))→ Blur}
Description: Crop image to feature all faces that are smiling and have eyes open, except the groom's face.
Dataset: Wedding

(8) {Union(Is(Object(bride)), Intersect(Is(Smiling), Is(EyesOpen))) → Blur}
Description: Crop image to feature the bride's face, plus faces that are smiling and have their eyes open.
Dataset: Wedding

(9) {Intersect(Complement(Is(Smiling)), Find(Is(Object(face)), (Is(Object(face)), GetAbove)) → Blur}
Description: Blur all faces in the back that are not smiling.
Dataset: Wedding

(10) {Union(Intersect(Is(Object(face)), Complement(Is(Smiling))), Is(BelowAge(18))) → Blur}
Description: Blur all faces that are not smiling or are under 18.
Dataset: Wedding

(11) {Union(Find(Is(Object(bride)), Is(Object(face)), GetRight), Is(Object(bride))) → Crop}
Crop image to feature just the bride's face and the face directly to her right.
Dataset: Wedding

(12) {Union(Is(Object(bride)), Find(Is(Object(bride)), Is(Object(groom)), GetAbove)) → Crop}
Description: Crop image to feature just the bride and the groom when he is behind her.
Dataset: Wedding
(13) {Intersect(Find(Is(Object(face)), Is(Object(face)), GetRight),
Find(Is(Object(face)), Is(Object(face)), GetLeft)) → Brighten}
Description: Brighten all faces except leftmost and rightmost face.
Dataset: Wedding
(14) {Find(Union(Is(Object(groom)), Is(Smiling), Is(EyesOpen)), Is(Object(person)), GetBelow) →
Sharpen}
Description: Sharpen the groom, and all smiling people and people with their eyes open.
Dataset: Wedding
(15) {Intersect(Find(Is(Object(face)), Is(Object(bride)), GetRight),
Find(Is(Object(face)), Is(Object(bride)),
GetLeft)) → Crop}
Description: Crop image to feature just bride when someone is to her left and right.
Dataset: Wedding
(16) {Union(Find(Is(Object(bride)), Is(Object(face)), GetRight), Find(Is(Is(Object(bride))),
FaceObject, GetLeft), Is(Object(bride)))
→ Crop}
Description: Crop image to feature just the bride and the people to her left and right.
Dataset: Wedding
(17) {Complement(Is(Object(car))) → Blur}
Description: Blur all objects except cars.
Dataset: City Streets
(18) {Filter(Is(Object(car)), Is(Object(face))) → Blur}
Description: Blur all faces in cars.
Dataset: City Streets
(19) {Filter(Is(Object(car)), Is(Object(text)) → Blur}
Description: Blur all text on cars.
Dataset: City Streets
(20) {Find(Is(Object(text)), Is(Object(car)), GetParents) → Blur}
Description: Blur all cars with text on them.
Dataset: City Streets
(21) {Union(Is(Object(cat)), Is(Object(face))) → Brighten}
Description: Brighten all faces and all cats.
Dataset: Cats
(22) {Union(Is(Object(cat)), Is(EyesOpen)) → Brighten}
Description: Brighten all faces with eyes open and all cats.
Dataset: Cats
(23) {Find(Is(Object(guitar)), Is(Object(face)), GetAbove) → Sharpen}
Description: Sharpen faces of people playing guitar.
Dataset: Festival
(24) {Find(Intersect(Is(Smiling), Is(EyesOpen)), Object(car),
GetParents) → Brighten}
Description: Brighten all people in cars who are smiling and have eyes open.
Dataset: City Streets

(25) {Union(Is(Object(car)), Is(Object(bicycle))) → Brighten}
Description: Brighten all cars and bicycles.
Dataset: City Streets
(26) {Find(Is(Object(person)), Object(bicycle), GetBelow) → Brighten}
Description: Brighten all bicycles that are being ridden.
Dataset: City Streets
(27) {Find(Is(Object(bicycle)), BelowAge(18), GetAbove) → Blur}
Description: Blur the faces of children riding bicycles.
Dataset: City Streets
(28) {Complement(Union(Is(Object(car)), Is(Object(bicycle)))) → Blackout}
Description: Blackout all objects except cars and bicycles.
Dataset: City Streets
(29) {Intersect(Is(Object(text)), Complement(Filter(Is(Object(car)), Is(Object(car))))) → Blackout}
Description: Blackout all text not on a car.
Dataset: City Streets
(30) {Union(Is(Object(bicycle)), Is(Object(car)), Is(Object(person))) → Brighten}
Description: Brighten all bicycles, cars, and people.
Dataset: City Streets
(31) {Intersect(Is(Object(face)),
Complement(Find(Is(Object(bicycle)), Is(Object(face)), GetAbove))) → Blur}
Description: Blur faces of people not riding bicycles.
Dataset: City Streets
(32) {Union(Is(Object(guitar)), Find(Is(Object(guitar)), Is(Object(face)), GetAbove)) → Brighten}
Description: Brighten all guitars and people playing guitar.
Dataset: Festival
(33) {Intersect(Is(Object(face)), Complement(Find(Is(Object(guitar)), Is(Object(face)), GetAbove)))
→ Blur}
Description: Blur faces of people not playing guitar.
Dataset: Festival
(34) {Intersect(Is(Object(bicycle)),
Complement(Find(Is(Object(person)), Object(bicycle), GetBelow))) → Sharpen}
Description: Sharpen bicycles that are not being ridden.
Dataset: City Streets
(35) {Intersect(Is(Object(bicycle)), Complement(Find(Is(BelowAge(18)), Is(Object(bicycle)), GetBelow))) → Sharpen}
Description: Sharpen all bicycles that are not ridden by a child.
Dataset: City Streets
(36) {Intersect(Is(Object(cat)),
Complement(Find(Is(Object(cat)),
Object(cat), GetBelow))) → Crop }
Description: Crop image to feature just topmost cat.
Dataset: Cats
(37) {Intersect(Find(Is(Object(cat)),
Object(cat), GetRight),
Find(Is(Object(cat)), Object(cat),
GetLeft)) → Brighten}

Description: Brighten cats that are between two other cats.
Dataset: Cats

E List of ImageSearch Benchmarks

(1) $\exists x.\exists y.\text{HasAttribute}(x, \texttt{car}) \wedge \text{HasAttribute}(y, \texttt{bicycle})$
Description: Images that contain a car and a bicycle.
Dataset: City Streets
(2) $\exists x.\exists y.\text{HasAttribute}(x, \texttt{guitar}) \wedge \text{HasAttribute}(y, \texttt{microphone})$
Description: Images that contain a guitar and a microphone.
Dataset: Festival
(3) $\forall x.\neg\text{HasAttribute}(x, \texttt{person})$
Description: Images that do not contain any people.
Dataset: Wedding
(4) $\exists x.\exists y.\text{HasAttribute}(x, \texttt{bride}) \wedge \text{HasAttribute}(y, \texttt{groom}) \wedge \text{HasRelation}(x, y, \texttt{left})$
Description: Images where the bride is to the left of the groom.
Dataset: Wedding
(5) $\exists x.\forall y.\text{HasAttribute}(x, \texttt{bride}) \wedge \neg\text{HasAttribute}(y, \texttt{groom})$
Description: Images that contain the bride and not the groom.
Dataset: Wedding
(6) $\exists x.\exists y.\text{HasAttribute}(x, \texttt{face}) \wedge \text{HasAttribute}(y, \texttt{glasses}) \wedge \text{HasRelation}(x, y, \texttt{contains})$
Description: Images with people wearing glasses. Dataset: Wedding
(7) $\exists x.\forall y.\text{HasAttribute}(x, \texttt{face}) \wedge (\text{HasAttribute}(y, \texttt{glasses}) \rightarrow \neg\text{HasRelation}(x, y, \texttt{contains}))$
Description: Images with people not wearing glasses. Dataset: Wedding
(8) $\exists x.\exists y.\text{HasAttribute}(x, \texttt{mouth\_open}) \wedge \text{HasAttribute}(y, \texttt{mouth\_open}) \wedge \text{HasRelation}(x, y, \texttt{next\_to})$
Description: Images with people talking.
Dataset: Wedding
(9) $\exists x.\exists y.\text{HasAttribute}(x, \texttt{mouth\_open}) \wedge \text{HasAttribute}(x, \texttt{smiling}) \wedge \text{HasAttribute}(y, \texttt{mouth\_open}) \wedge$
$\text{HasAttribute}(y, \texttt{smiling}) \wedge \text{HasRelation}(x, y, \texttt{next\_to})$
Description: Images with people laughing.
Dataset: Wedding
(10) $\exists x.\exists y.\text{HasAttribute}(x, \texttt{tie}) \wedge \text{HasAttribute}(y, \texttt{face}) \wedge \text{HasRelation}(x, y, \texttt{below})$
Description: Images with people wearing ties.
Dataset: Wedding
(11) $\exists x.\text{HasAttribute}(x, \texttt{suit}) \vee \text{HasAttribute}(x, \texttt{tie}) \vee \text{HasAttribute}(x, \texttt{gown})$
Description: Images with formal attire.
Dataset: Wedding
(12) $\exists x.\text{HasAttribute}(x, \texttt{bride}) \wedge \text{HasAttribute}(x, \texttt{smiling})$
Description: Images where the bride is smiling.
Dataset: Wedding
(13) $\exists x.\exists y.\text{HasAttribute}(x, \texttt{guitar}) \wedge \text{HasAttribute}(y, \texttt{face}) \wedge \text{HasRelation}(x, y, \texttt{below})$
Description: Images with guitar players.
Dataset: Festival
(14) $\exists x.\exists y.\text{HasAttribute}(x, \texttt{microphone}) \wedge \text{HasAttribute}(y, \texttt{face}) \wedge \text{HasRelation}(x, y, \texttt{below})$
Description: Images with singers.
Dataset: Festival
(15) $\exists x.\exists y.\exists z.\text{HasAttribute}(x, \texttt{microphone}) \wedge \text{HasAttribute}(y, \texttt{face}) \wedge \text{HasAttribute}(z, \texttt{guitar}) \wedge$
$\text{HasRelation}(x, y, \texttt{below}) \wedge \text{HasRelation}(z, y, \texttt{below})$

Description: Images with people singing and playing guitar.
Dataset: Festival

(16) $\exists x.\exists y.\text{HasAttribute}(x, \texttt{microphone}) \wedge \text{HasAttribute}(y, \texttt{face}) \wedge \text{HasRelation}(x, y, \texttt{below})$
Description: Images with smiling performers.
Dataset: Festival

(17) $\exists x.\text{HasAttribute}(x, \texttt{speaker}) \vee \text{HasAttribute}(x, \texttt{microphone}) \vee \text{HasAttribute}(x, \texttt{guitar})$
Description: Images with stage equipment.
Dataset: Festival

(18) $\exists x.\exists y.\text{HasAttribute}(x, \texttt{bicycle}) \wedge \text{HasAttribute}(y, \texttt{face}) \wedge \text{HasRelation}(x, y\texttt{below})$
Description: Images with cyclists.
Dataset: City Streets

(19) $\exists x.\forall y.\text{HasAttribute}(x, \texttt{bicycle}) \wedge \neg\text{HasAttribute}(y, \texttt{car})$
Description: Images with bicycles but not cars.
Dataset: City Streets

(20) $\exists x.\exists y.\text{HasAttribute}(x, \texttt{bicycle}) \wedge \text{HasAttribute}(y, \texttt{smiling}) \wedge \text{HasRelation}(x, y\texttt{below})$
Description: Images with happy cyclists.
Dataset: City Streets

(21) $\exists x.\exists y.\exists z.\text{HasAttribute}(x, \texttt{bicycle}) \wedge \text{HasAttribute}(y, \texttt{face}) \wedge \text{HasAttribute}(y, \texttt{helmet}) \wedge \text{HasRelation}(x, y, \texttt{below}) \wedge \text{HasRelation}(y, z, \texttt{below})$
Description: Images with cyclists wearing helmets.
Dataset: City Streets

(22) $\exists x.\exists y.\text{HasAttribute}(x, \texttt{car}) \wedge \text{HasAttribute}(y, \texttt{face}) \wedge \text{HasRelation}(x, y, \texttt{contains})$
Description: Images with people driving cars.
Dataset: City Streets

(23) $\exists x.\forall y.\text{HasAttribute}(x, \texttt{face}) \wedge (\text{HasAttribute}(y, \texttt{car}) \rightarrow \neg\text{HasRelation}(y, x, \texttt{contains})$
Description: Images with people not driving cars.
Dataset: City Streets

(24) $\exists x.\text{HasAttribute}(x, \texttt{bicycle}) \vee \text{HasAttribute}(x, \texttt{car})$
Description: Images with modes of transportation. Dataset: City Streets

(25) $\exists x.\exists y.\text{HasAttribute}(x, \texttt{hat}) \wedge \text{HasAttribute}(y, \texttt{face}) \wedge \text{HasRelation}(x, y, \text{above})$
Description: Images with people wearing hats.
Dataset: City Streets

F List of VisArith Benchmarks

(1) $\lambda\, l.$`fold inc 0 (filter (curry` $\leq$ `5) (map (curry plus 1) (map toDigit` l`)))`
Counts how many test scores are still low (below 5) after adding an extra credit point.

(2) $\lambda\, l.$`fold max 0 (map (curry plus (toDigit (head` l`)) (filter (curry` $\leq$ `8) (map toDigit (tail` l`)))))`
Calculates the highest score after x extra credit points are added to scores below 8.

(3) $\lambda l.$`fold plus 0 (map (curry plus (toDigit (head` l`))) (filter (curry` $\leq$ `6) (map toDigit (tail` l`))))`
For cheap products being sold at a store, calculate the total revenue from selling the products after increasing the price by x dollars.

(4) $\lambda\, l.$`fold inc 0 (filter (curry plus (toDigit (head` l`))) (map (curry mult 2) (map toDigit (tail` l`))))`
After doubling the amount of every product in the inventory, count how many have more than x units in stock, where x is the required minimum.

(5) $\lambda\, l$.fold max 0 (map (curry plus (toDigit (head l))) (filter (curry $\leq$ 2) (map toDigit (tail l))))
Researchers at the North Pole took notes of readings from a thermometer that is inaccurate in extreme cold. For temperatures below 3 fahrenheit corrects the false temperature readings by increasing them by x. Then reports the highest temperature at which the sensor starts failing.
(6) $\lambda\, l$.fold plus 0 (filter (curry $\geq$ (toDigit (head l))) (map toDigit (tail l)))
Sum of digits greater than or equal to x.
(7) $\lambda\, l$.fold prod 1 (filter (curry $\geq$ (toDigit (head l))) (map toDigit (tail l)))
Product of digits less than x after multiplying by 3.
(8) $\lambda\, l$.fold inc 0 (filter (curry $\leq$ 8) (filter (curry $\geq$ 8) (map toDigit (tail l))))
Computes the number of occurences of a single digit (8) in the list.
(9) $\lambda\, l$.fold plus (toDigit (head l))(filter (curry $\geq$ (toDigit (head l))) (map toDigit (tail l)))
Counts the total sum of donations exceeding a threshold x, plus an additional donation of x.
(10) $\lambda\, l$.fold plus 0 (map (curry prod 2) (map toDigit (tail l)))
Calculates the total sum of donations that have been matched.
(11) $\lambda\, l$.fold max (toDigit (head l))(filter (curry $\geq$ 5) (map toDigit (tail l)))
Determines the maximum age of children in a day care over age 4. If there are no children over age 4, default to x.
(12) $\lambda\, l$.fold plus 0 (map (curry plus (toDigit (head l))) (map toDigit (tail l)))
Calculates the inventory count after restocking.
(13) $\lambda\, l$.fold inc 0 (filter (curry $\leq$ (toDigit (head l))) (map toDigit (tail l)))
Counts the number of retail transactions below a limit x.
(14) $\lambda\, l$.fold prod 1 (map (curry prod 7) (map toDigit (tail l)))
Computes the product of a list of numbers after multiplying them by 7.
(15) $\lambda\, l$.fold inc 0 (filter (curry $\geq$ 4) (filter (curry $\leq$ 8) (map toDigit (tail l))))
Counts the number of participants in a study between the ages of 4 and 8.
(16) $\lambda\, l$.fold max 0 (filter (curry $\leq$ 2) (map toDigit (tail l)))
Finds the maximum sub-2k dollar expenditure on a balance sheet.
(17) $\lambda\, l$.fold plus 0 (filter (curry $\geq$ 8) (map toDigit (tail l)))
Finds the sum of expenditures over $8k on a balance sheet.
(18) $\lambda\, l$.fold map 0 (map (curry prod 2) (map toDigit (tail l)))
Finds the size of the largest class if every class size were to double.
(19) $\lambda\, l$.fold max 0 (map (curry prod 2) (filter (curry $\geq$ 5) (map toDigit (tail l))))
Finds the size of the largest class, if every class with less than 5 students dissolved and every other class size doubled.
(20) $\lambda\, l$.fold inc 0 (filter (curry $\geq$ 1) (map toDigit (tail l)))
Counts the number number of non-zero digits in the list.
(21) $\lambda\, l$.fold prod 1 (map toDigit (tail l))
Calculates cumulative compound interest by years.
(22) $\lambda\, l$.fold max 0 (map toDigit (tail l))
Calculates maximum score on a midterm.
(23) $\lambda\, l$.fold inc 0 (filter (curry $\geq$ (toDigit (head l))) (map toDigit (tail l)))
Calculates number of people with a passing grade.
(24) $\lambda\, l$.fold inc 0 (filter (curry $\leq$ 0) (map toDigit (tail l)))
Counts the number of 0's in the list.

(25) $\lambda\, l$.fold max 0 (filter (curry $\geq$ (map toDigit (tail l))))
Finds the maximum among elements equaling at least 5, defaulting to 0 if there are no such elements.
(26) $\lambda\, l$.fold max 0 (filter (curry $\leq$ (toDigit (head l))) (map toDigit (tail l)))
Finds the highest failing grade on a test.
(27) $\lambda\, l$.fold plus 0 (map (curry prod 6) (map toDigit (tail l)))
Computes the total sales revenue by multiplying the quantity of items sold by unit price 6, and summing the results.
(28) $\lambda\, l$.fold plus 0 (filter (curry $\geq$ 2) (map toDigit (tail l)))
Counts the number of batteries in boxes containing at least 2 batteries.
(29) $\lambda\, l$.fold inc 0 (filter (curry $\geq$ (toDigit (head l))) (map (curry plus 4) (map toDigit (tail l))))
Determines the number of students who would pass the course after adding 4 bonus points.
(30) $\lambda\, l$.fold inc 0 (filter (curry $\geq$ (toDigit (head l))) (filter (curry $\geq$ 7) (map toDigit (tail l))))
Determines the number of objects within a size range suitable for a robot's gripper.
(31) $\lambda\, l$.fold inc 0 (filter (curry $\geq$ 2) (map toDigit (tail l)))
Counts the number of conference attendees who have more than 1 dietary restriction.
(32) $\lambda\, l$.fold plus 0 (map (curry prod (toDigit (head l))) (map toDigit (tail l)))
Calculates the total weekly egg budget, where each person needs a specific number of eggs perday, and each egg costs x dollars.
(33) $\lambda\, l$.fold max 0 (map (curry plus (toDigit (head l))) (map toDigit (tail l)))
Identifies the most expensive neutral oil in the grocery store, adding x dollars of tax.
(34) $\lambda\, l$.fold inc 0 (filter (curry $\geq$ 4) (map toDigit (tail l)))
Counts the number of restaurants with ratings of at least 4 from a list of restaurant ratings.
(35) $\lambda\, l$.fold max 0 (map (curry plus 5) (map toDigit (tail l)))
Identifies the microwave recipe with the longest cooking time, adding 5 minutes for prep.
(36) $\lambda\, l$.fold inc 0 (filter (curry $\geq$ 5) (map toDigit (tail l)))
Counts the total number of donations exceeding $5.
(37) $\lambda\, l$.fold inc 0 (filter (curry $\geq$ 3) (map toDigit (tail l)))
In a pool of PhD applications, counts how many have a GPA of at least 3.
(38) $\lambda\, l$.fold inc 0 (filter (curry $\leq$ (toDigit (head l))) (map (curry plus 2) (map toDigit (tail l))))
Counts how many students cannot get a C grade (where the cutoff is x, after adding 2 bonus points.
(39) $\lambda\, l$.fold inc 0 (filter (curry $\leq$ (toDigit (head l))) (map (curry plus 1) (map toDigit (tail l))))
Counts how many reviewers have ratings below x, if each score were increased by 1.
(40) $\lambda\, l$.fold plus 1 (map (curry max 8) (filter (curry $\geq$ 4) (map toDigit (tail l))))
Calculates the sum of the prices of objects in a store, when all the products less than $4 are removed, while remaining prices are increased to at least $8.
(41) $\lambda\, l$.fold plus 0 (filter (curry $\leq$ (toDigit (head l))) (map toDigit (tail l)))
Calculates the total score among football teams scoring less than or equal to x points.
(42) $\lambda\, l$.fold prod 1 (map (curry max 1) (map toDigit (tail l)))
Finds the product of a list, with 0's replaced with 1's.
(43) $\lambda\, l$.fold prod 1 (filter (curry $\geq$ 2) (filter (curry $\leq$ 4) (map toDigit (tail l))))
Finds the product of all numbers in a list between 1 and 5 exclusive.

(44) $\lambda\, l$.fold plus 0 (filter (curry prod 3) (map toDigit (tail l)))
Finds the total score in the last round of Family Feud, where points are worth triple.
(45) $\lambda\, l$.fold inc 0 (filter (curry $\geq$ 1) (filter (curry $\leq$ 5) (map toDigit (tail l))))
Finds the number of non-zero race times less than or equal to 5.
(46) $\lambda\, l$.fold plus 0 (filter (curry $\geq$ 3) (map toDigit (tail l)))
Given a list of purchases at a cash register, gets the sum of transactions over $2.
(47) $\lambda\, l$.fold inc 0 (filter (curry $\geq$ (toDigit (head l))) (map (curry plus1 (map toDigit (tail l)))))
Count how many applicants have a GPA of at least $x - 1$.
(48) $\lambda\, l$.fold plus 0 (map (curry plus 2) (map (curry prod (toDigit (head l))) (map toDigit (tail l))))
Computes the total cost of items in a box, where the cost per item is x, and the flat cost of the box is $2.
(49) $\lambda\, l$.fold max 0 (filter (curry $\leq$ 7) (map toDigit (tail l)))
Given a list of people wanting various amounts of rice under 8kg, finds the one who requested the most rice.
(50) $\lambda\, l$.fold max 0 (filter (curry $\leq$ (toDigit (head l))) (map (curry prod 5) (map toDigit (tail l))))
Finds for the max weight of an item, such that you can put 5 of them on a lift with a maximum weight of x.

Received 2025-03-25; accepted 2025-08-12