Visual Programming: Compositional visual reasoning without training

Tanmay Gupta, Aniruddha Kembhavi
PRIOR @ Allen Institute for AI
https://prior.allenai.org/projects/visprog

Figure 1. VisPROG is a modular and interpretable neuro-symbolic system for compositional visual reasoning. Given a few examples of natural language instructions and the desired high-level programs, VisPROG generates a program for any new instruction using in-context learning in GPT-3 and then executes the program on the input image(s) to obtain the prediction. VisPROG also summarizes the intermediate outputs into an interpretable visual rationale (Fig. 4). We demonstrate VisPROG on tasks that require composing a diverse set of modules for image understanding and manipulation, knowledge retrieval, and arithmetic and logical operations.

Abstract

We present VisPROG, a neuro-symbolic approach to solving complex and compositional visual tasks given natural language instructions. VisPROG avoids the need for any task-specific training. Instead, it uses the in-context learning ability of large language models to generate python-like modular programs, which are then executed to get both the solution and a comprehensive and interpretable rationale. Each line of the generated program may invoke one of several off-the-shelf computer vision models, image processing subroutines, or python functions to produce intermediate outputs that may be consumed by subsequent parts of the program. We demonstrate the flexibility of VisPROG on 4 diverse tasks - compositional visual question answering, zero-shot reasoning on image pairs, factual knowledge object tagging, and language-guided image editing. We believe neuro-symbolic approaches like VisPROG are an exciting avenue to easily and effectively expand the scope of AI systems to serve the long tail of complex tasks that people may wish to perform.
1. Introduction

The pursuit of general purpose AI systems has lead to the development of capable end-to-end trainable models [1, 5, 8, 13, 17, 22, 24], many of which aspire to provide a simple natural language interface for a user to interact with the model. The predominant approach to building these systems has been massive-scale unsupervised pretraining followed by supervised multitask training. However, this approach requires a well curated dataset for each task that makes it challenging to scale to the infinitely long tail of complex tasks we would eventually like these systems to perform. In this work, we explore the use of large language models to tackle the long tail of complex tasks by decomposing these tasks described in natural language into simpler steps that may be handled by specialized end-to-end trained models or other programs.

Imagine instructing a vision system to “Tag the 7 main characters on the TV show Big Bang Theory in this image.” To perform this task, the system first needs to understand the intent of the instruction and then perform a sequence of steps - detect the faces, retrieve list of main characters on Big Bang Theory from a knowledge base, classify faces using the list of characters, and tag the image with recognized character’s faces and names. While different vision and language systems exist to perform each of these steps, executing this task described in natural language is beyond the scope of end-to-end trained systems.

We introduce VisPROG which inputs visual data (a single image or a set of images) along with a natural language instruction, generates a sequence of steps, a visual program if you will, and then executes these steps to produce the desired output. Each line in a visual program invokes one among a wide range of modules currently supported by the system. Modules may be off-the-shelf computer vision models, language models, image processing subroutines in OpenCV [4], or arithmetic and logical operators. Modules consume inputs that are produced by executing previous lines of code and output intermediate results that can be consumed downstream. In the example above, the visual program generated by VisPROG invokes a face detector [16], GPT-3 [5] as a knowledge retrieval system, and CLIP [20] as an open-vocabulary image classifier to produce the desired output (see Fig. 1).

VisPROG improves upon previous methods for generating and executing programs for vision applications. For the visual question answering (VQA) task, Neural Module Networks (NMN) [2, 9, 10, 12] compose a question-specific, end-to-end trainable network from specialized, differentiable neural modules. These approaches either use brittle, off-the-shelf semantic parsers to deterministically compute the layout of modules, or learn a layout generator through weak answer supervision via REINFORCE [30]. In contrast, VisPROG uses a powerful language model (GPT-3) and a small number of in-context examples to create complex programs without requiring any training.1 Programs created by VisPROG also use a higher-level of abstraction than NMNs and invoke trained state-of-the-art models and non-neural python subroutines (Fig. 2). These advantages make VisPROG an easy-to-use, performant, and modular neuro-symbolic system.

VisPROG is also highly interpretable. First, VisPROG produces easy-to-understand programs which a user can verify for logical correctness. Second, by breaking down the prediction into simple steps, VisPROG allows a user to inspect the outputs of intermediate steps to diagnose errors and if required, intervene in the reasoning process. Altogether, an executed program with intermediate step results (e.g. text, bounding boxes, segmentation masks, generated images, etc.) linked together to depict the flow of information serves as a visual rationale for the prediction.

To demonstrate its flexibility, we use VisPROG for 4 different tasks that share some common skills (e.g. for image parsing) while also requiring some degree of specialized reasoning and visual manipulation capabilities. These tasks are - (i) compositional visual question answering; (ii) zero-shot natural language visual reasoning (NLVR) on image pairs; (iii) factual knowledge object tagging from natural language instructions; and (iv) language-guided image editing. We emphasize that neither the language model nor any of the modules are finetuned in any way. Adapting VisPROG to any task as simple as providing a few in-context examples consisting of natural language instructions and the corresponding programs. While easy to use, VisPROG shows an impressive gain of 2.7 points over a base VQA model on the compositional VQA task, strong zero-shot accuracy of 62.4% on NLVR without ever training on image pairs, and delightful qualitative and quantitative results on knowledge tagging and image editing tasks.

---
1We use “training” to refer to gradient-based learning to differentiate it from in-context learning which only involves a feedforward pass.
Our key contributions include - (i) VISPROG - a system that uses the in-context learning ability of a language model to generate visual programs from natural language instructions for compositional visual tasks (Sec. 3); (ii) demonstrating the flexibility of VISPROG on complex visual tasks such as factual knowledge object tagging and language guided image editing (Secs. 4.3 and 4.4) that have eluded or seen limited success with a single end-to-end model; and (iii) producing visual rationales for these tasks and showing their utility for error analysis and user-driven instruction tuning to improve VISPROG’s performance significantly (Sec. 5.3).

2. Related Work

Neuro-symbolic approaches have seen renewed momentum owing to the incredible understanding, generation, and in-context learning capabilities of large language models (LLMs). We now discuss previous program generation and execution approaches for visual tasks, recent work in using LLMs for vision, and advances in reasoning methods for language tasks.

Program generation and execution for visual tasks. Neural module networks (NMN) [2] pioneered modular and compositional approaches for the visual question answering (VQA) task. NMNs compose neural modules into an end-to-end differentiable network. While early attempts use off-the-shelf parsers [2], recent methods [9, 10, 12] learn the layout generation model jointly with the neural modules using REINFORCE [30] and weak answer supervision. While similar in spirit to NMNs, VISPROG has several advantages over NMNs. First, VISPROG generates high-level programs that invoke trained state-of-the-art neural modules and other python functions at intermediate steps as opposed to generating end-to-end neural networks. This makes it easy to incorporate symbolic, non-differentiable modules. Second, VISPROG leverages the in-context learning ability of LLMs [5] to generate programs by prompting the LLM (GPT-3) with a natural language instruction (or a visual question or a statement to be verified) along with a few examples of similar instructions and their corresponding programs thereby removing the need to train specialized program generators for each task.

LLMs for visual tasks. LLMs and in-context learning have been applied to visual tasks. PICa [31] uses LLMs for a knowledge-based VQA [18] task. PICa represents the visual information in images as text via captions, objects, and attributes and feeds this textual representation to GPT-3 along with the question and in-context examples to directly generate the answer. Socratic models (SMs) [33], compose pretrained models from different modalities such as language (BERT [7], GPT-2 [21]), vision-language (CLIP [20]), and audio-language (mSLAM [3]), to perform a number of zero-shot tasks, including image captioning, video-to-text retrieval, and robot planning. However, in SMs the composition is pre-determined and fixed for each task. In contrast, VISPROG determines how to compose models for each instance by generating programs based on the instruction, question, or statement. We demonstrate VISPROG’s ability to handle complex instructions that involve diverse capabilities (20 modules) and varied input (text, image, and image pairs), intermediate (text, image, bounding boxes, segmentation masks), and output modalities (text and images). Similar to VISPROG, ProgPrompt [26] is a concurrent work that demonstrates the ability of LLMs to generate python-like situated robot action plans from natural language instructions. While ProgPrompt modules (such as “find” or “grab”) take strings (typically object names) as input, VISPROG programs are more general. In each step in a VISPROG program, a module could accept multiple arguments including strings, numbers, arithmetic and logical expressions, or arbitrary python objects (such as list() or dict() instances containing bounding boxes or segmentation masks) produced by previous steps.

Reasoning via Prompting in NLP. There is a growing body of literature on using LLMs for language reasoning tasks via prompting. Chain-of-Thought (CoT) prompting [29], where a language model is prompted with in-context examples of inputs, chain-of-thought rationales (a series of intermediate reasoning steps), and outputs, has shown impressive abilities for solving math reasoning problems. While CoT relies on the ability of LLMs to both generate a reasoning path and execute it, approaches similar to VISPROG have been applied to language tasks, where a decomposer prompt [14] is used first to generate a sequence of sub-tasks which are then handled by sub-task handlers.

3. Visual Programming

Over the last few years, the AI community has produced high-performance, task-specific models for many vision and language tasks such as object detection, segmentation, VQA, captioning, and text-to-image generation. While each of these models solves a well-defined but narrow problem, the tasks we usually want to solve in the real world are often broader and loosely defined.

To solve such practical tasks, one has to either collect a new task-specific dataset, which can be expensive, or meticulously compose a program that invokes multiple neural models, image processing subroutines (e.g. image resizing, cropping, filtering, and colorspace conversions), and other computation (e.g. database lookup, or arithmetic and logical operations). Manually creating these programs for the infinitely long tail of complex tasks we encounter daily not only requires programming expertise but is also slow, labor intensive, and ultimately insufficient to cover the space of all tasks. What if, we could describe the task in natural language and have an AI system generate and execute the...
corresponding visual program without any training?

**Large language models for visual programming.** Large language models such as GPT-3 have shown a remarkable ability to generalize to new samples for a task having seen a handful of input and output demonstrations in-context. For example, prompting GPT-3 with two English-to-French translation examples and a new English phrase

```
good morning --> bonjour
```

produces the French translation “bonsoir”. Note that we did not have to finetune GPT-3 to perform the task of translation on the third phrase. VisPROG uses this in-context learning ability of GPT-3 to output visual programs for natural language instructions.

Similar to English and French translation pairs in the example above, we prompt GPT-3 with pairs of instructions and the desired high-level program. Fig. 3 shows such a prompt for an image editing task. The programs in the in-context examples are manually written and can typically be constructed without an accompanying image. Each line of a VisPROG program, or a **program step**, consists of the name of a **module**, module’s input argument names and their values, and an output variable name. VisPROG programs often use output variables from past steps as inputs to future steps. We use descriptive module names (e.g. “Select”, “ColorPop”, “Replace”), argument names (e.g. “image”, “object”, “query”), and variable names (e.g. “IMAGE”, “OBJ”) to allow GPT-3 to understand the input and output type, and function of each module. During execution the output variables may be used to store arbitrary data types. For instance “OBJ”s are list of objects in the image, with mask, bounding box, and text (e.g. category name) associated with each object.

These in-context examples are fed into GPT-3 along with a new natural language instruction. Without observing the image or its content, VisPROG generates a program (bottom of Fig. 3) that can be executed on the input image(s) to perform the described task.

![VisPROG Program Generation](image)

**Figure 3. Program generation in VisPROG.**

**Code 1. Implementation of a VisPROG module.**

```
class VisProGModule:
    def __init__(self):
        # load a trained model; move to GPU

    def html(self, inputs: List, output: Any):  # visual summary of the step computation
        # return an html string visualizing step I/O
        step_html = self.html(inputs, output)
        return step_html

    def parse(self, step: str):  # parse step and return list of input values/variable names
        # and output variable name
        inputs, input_var_names, output_var_name = self.parse(step)
        return inputs, input_var_names, output_var_name

    def execute(self, step: str, state: Dict):  # perform computation using the loaded model
        # get values of input variables from state
        output = some_computation(inputs)
        return output

    def update_state(self, state):  # update state
        return state
```

**Modules.** VisPROG currently supports 20 modules (Fig. 2) for enabling capabilities such as image understanding, image manipulation (including generation), knowledge retrieval, and performing arithmetic and logical operations. In VisPROG, each module is implemented as a Python class (Code. 1) that has methods to: (i) **parse** the line to extract the input argument names and values, and the output variable name; (ii) **execute** the necessary computation that may involve trained neural models and update the program state with the output variable name and value; and (iii) summarize the step’s computation visually using html (used later to create a visual rationale). Adding new modules to VisPROG simply requires implementing and registering a module class, while the execution of the programs using this module is handled automatically by the VisPROG interpreter, which is described next.

**Program Execution.** The program execution is handled by an **interpreter**. The interpreter initializes the program state (a dictionary mapping variables names to their values) with the inputs, and steps through the program line-by-line while invoking the correct module with the inputs specified in that line. After executing each step, the program state is updated with the name and value of the step’s output.

**Visual Rationale.** In addition to performing the necessary computation, each module class also implements a method called html() to visually summarize the inputs and outputs of the module in an HTML snippet. The interpreter simply stitches the HTML summary of all program steps into a visual rationale (Fig. 4) that can be used to analyze the logical correctness of the program as well as inspect the in-
We evaluate V\textsubscript{P}=2 0\textsuperscript{set with desired V\textsubscript{P}=5\textsuperscript{P} to use a single-image
testdev\textsuperscript{(}) and Visual rationales generated by V\textsubscript{P}\textsubscript{P} is compositional by construction which makes
for more details.
intermediate outputs. The visual rationales also enable users
to understand reasons for failure and tweak the natural lan-
guage instructions minimally to improve performance. See
Sec. 5.3 for more details.

4. Tasks

VisPROG provides a flexible framework that can be ap-
plicated to a diverse range of complex visual tasks. We eval-
uate VisPROG on 4 tasks that require capabilities ranging
from spatial reasoning, reasoning about multiple images,
knowledge retrieval, and image generation and manipula-
tion. Fig. 5 summarizes the inputs, outputs, and modules
used for these tasks. We now describe these tasks, their
evaluation settings, and the choice of in-context examples.

4.1. Compositional Visual Question Answering

VisPROG is compositional by construction which makes
it suitable for the compositional, multi-step visual ques-
tion answering task: GQA [11]. Modules for the GQA
task include those for open vocabulary localization, a VQA
module, functions for cropping image regions given bound-
ning box co-ordinates or spatial prepositions (such as above,
left, etc.), module to count boxes, and a module to evaluate
Python expressions. For example, consider the question:

"Is the small truck to the left or to the right of the people
that are wearing helmets?". VisPROG first localizes "peo-
ple wearing helmets", crops the region to the left (or right)
of these people, checks if there is a "small truck" on that
side, and return “left” if so and “right” otherwise. VisPROG
uses the question answering module based on ViLT [15],
but instead of simply passing the complex original question
to ViLT, VisPROG invokes it for simpler tasks like identi-
fying the contents within an image patch. As a result, our
resulting VisPROG for GQA is not only more interpretable
than ViLT but also more accurate (Tab. 1). Alternatively,
one could completely eliminate the need for a QA model
like ViLT and use other systems like CLIP and object de-
dectors, but we leave that for future investigation.

Evaluation. In order to limit the money spent on generating
programs with GPT-3, we create a subset of GQA for eval-
uation. Each question in GQA is annotated with a question
type. To evaluate on a diverse set of question types (~ 100
detailed types), we randomly sample up to k samples per
question type from the balanced \textit{val} (k = 5) and \textit{testdev}
(k = 20) sets.

Prompts. We manually annotate 31 random questions from
the balanced \textit{train} set with desired VisPROG programs. An-
donating questions with programs is easy and requires writ-
ing down the chain of reasoning required to answer that par-
ticular question. We provide a smaller subset of in context
eamples to GPT-3, randomly sampled from this list to re-
duce the cost of answering each GQA question.

4.2. Zero-Shot Reasoning on Image Pairs

VQA models are trained to answer questions about a sin-
gle image. In practice, one might require a system to an-
swer questions about a collection of images. For example,
a user may ask a system to parse their vacation photo album
and answer the question: “Which landmark did we visit, the
day after we saw the Eiffel Tower?” Instead of assembling
an expensive dataset and training a multi-image model, we
demonstrate the ability of VisPROG to use a single-image
VQA system to solve a task involving multiple images with-
out training on multi-image examples.

We showcase this ability on the NLVRv2 [27] bench-
mark, which involves verifying statements about image
pairs. Typically, tackling the NLVRv2 challenge requires training custom architectures that take image pairs as input on NLVRv2’s train set. Instead, VISPROG achieves this by decomposing a complex statement into simpler questions about individual images and a python expression involving arithmetic and logical operators and answers to the image-level questions. The VQA model ViLT-VQA is used to get image-level answers, and the python expression is evaluated to verify the statement.

**Evaluation.** We create a small validation set by sampling 250 random samples from the NLVRv2 dev set to guide prompt selection, and test generalization on NLVRv2’s full public test set.

**Prompts.** We sample and annotate VISPROG programs for 16 random statements in the NLVRv2 train set. Since some of these examples are redundant (similar program structure) we also create a curated subset of 12 examples by removing 4 redundant ones.

### 4.3. Factual Knowledge Object Tagging

We often want to identify people and objects in images whose names are unknown to us. For instance, we might want to identify celebrities, politicians, characters in TV shows, flags of countries, logos of corporations, popular cars and their manufacturers, species of organisms, and so on. Solving this task requires not only localizing people, faces, and objects but also looking up factual knowledge in an external knowledge base to construct a set of categories for classification, such as names of the characters on a TV show. We refer to this task as Factual Knowledge Object Tagging or Knowledge Tagging for short.

For solving Knowledge Tagging, VISPROG uses GPT-3 as an implicit knowledge base that can be queried with natural language prompts such as “List the main characters on the TV show Big Bang Theory separated by commas.” This generated category list can then be used by a CLIP image classification module that classifies image regions produced by localization and face detection modules. VISPROG’s program generator automatically determines whether to use a face detector or an open-vocabulary localizer depending on the context in the natural language instruction. VISPROG also estimates the maximum size of the category list retrieved. For instance, “Tag the logos of the top 5 german car companies” generates a list of 5 categories, while “Tag the logos of german car companies” produces a list of arbitrary length determined by GPT-3 with a cut-off at 20. This allows users to easily control the noise in the classification process by tweaking their instructions.

**Evaluation.** To evaluate VISPROG on this task, we annotate 100 tagging instructions across 46 images that require external knowledge to tag 253 object instances including personalities across pop culture, politics, sports, and art, as well as a varieties of objects (e.g. cars, flags, fruits, appliances, furniture etc.). For each instruction, we measure both localization and tagging performance via precision (fraction of predicted boxes that are correct) and recall (fraction of ground truth objects that are correctly predicted). Tagging metrics require both the predicted bounding box and the associated tag or class label to be correct, while localization ignores the tag. To determine localization correctness, we use an IoU threshold of 0.5. We summarize localization and tagging performance by F1 scores (harmonic mean of the average precision and recall across instructions).

**Prompts.** We create 14 in-context examples for this task. Note that the instructions for these examples were hallucinated i.e. no images were associated with these examples.

### 4.4. Image Editing with Natural Language

Text to image generation has made impressive strides over the last few years with models like DALL-E [23], Parti [32], and Stable Diffusion [25]. However, it is still beyond the capability of these models to handle prompts like “Hide the face of Daniel Craig with ‘p’” (de-identification or privacy preservation), or “Create a color pop of Daniel Craig and blur the background” (object highlighting) even though these are relatively simple to achieve programmatically using a combination of face detection, segmentation and image processing modules. Achieving a sophisticated edit such as “Replace Barack Obama with Barack Obama wearing sunglasses” (object replacement), first requires identifying the object of interest, generating a mask of the object to be replaced and then invoking an image inpainting model (we use Stable Diffusion) with the original image, mask specifying the pixels to replace, and a description of the new pixels to generate at that location. VISPROG, when equipped with the necessary modules and example programs, can handle very complex instructions with ease.

**Evaluation.** To test VISPROG on the image editing instructions for de-identification, object highlighting, and object replacement, we collect 107 instructions across 65 images. We manually score the predictions for correctness and report accuracy. Note that we do not penalize visual artifacts for the object replacement sub-task which uses Stable Diffusion as long as the generated image is semantically correct.

**Prompts.** Similar to knowledge tagging, we create 10 in-context examples for this task with no associated images.

### 5. Experiments and Analysis

Our experiments evaluate the effect of number of prompts on GQA and NLVR performance (Sec. 5.1), generalization of VISPROG on the four tasks comparing various prompting strategies (Sec. 5.2), analyze the sources of error for each task (Fig. 9), and study the utility of visual rationales for diagnosing errors and improving VISPROG’s performance through instruction tuning (Sec. 5.3).
5.1. Effect of prompt size

Fig. 6 shows that validation performance increases progressively with the number of in-context examples used in the prompts for both GQA and NLVR. Each run randomly selects a subset of the annotated in-context examples based on a random seed. We also find that majority voting across the random seeds leads to consistently better performance than the average performance across runs. This is consistent with findings in Chain-of-Thought [29] reasoning literature for math reasoning problems [28]. On NLVR, the performance of VISPROG saturates with fewer prompts than GQA. We believe this is because NLVRv2 programs require fewer modules and hence fewer demonstrations for using those modules than GQA.

5.2. Generalization

GQA. In Tab. 1 we evaluate different prompting strategies on the GQA testdev set. For the largest prompt size evaluated on the val set (24 in-context examples), we compare the random strategy consisting of the VISPROG’s best prompt chosen amongst 5 runs on the validation set (each run randomly samples in-context examples from 31 annotated examples) and the majority voting strategy which takes maximum consensus predictions for each question across 5 runs. While “random” prompts only slightly outperform VILT-VQA, voting leads to a significant gain of 2.7 points. This is because voting across multiple runs, each with a different set of in-context examples, effectively increases the total number of in-context examples seen for each prediction. We also evaluate a manually curated prompt consisting of 20 examples - 16 from the 31 annotated examples, and 4 additional hallucinated examples meant to provide a better coverage for failure cases observed in the validation set. The curated prompt performs just as well as the voting strategy while using 5× less compute, highlighting the promise of prompt engineering.

NLVR. Tab. 2 shows performance of VISPROG on the NLVRv2 test set and compares random, voting, and curated prompting strategies as done with GQA. While VISPROG performs the NLVR task zero-shot without ever training on image pairs, we report VILT-NLVR, a VILT model finetuned on NLVRv2 as an upper bound on performance. While several points behind the upper bound, VISPROG shows strong zero-shot performance using only a single-image VQA model for image understanding, and an LLM for reasoning. Note that, VISPROG uses VILT-VQA for its VQA module which is trained on VQAv2 a single image question answer task, but not NLVRv2.

Knowledge Tagging. Tab. 3 shows localization and tagging performance for the Knowledge Tagging task. All instructions for this task not only require open vocabulary localization but also querying a knowledge base to fetch the categories to tag localized objects with. This makes it an impossible task for object detectors alone. With the original instructions, VISPROG achieves an impressive 63.7% F1 score for tagging, which involves both correctly localizing and naming the objects, and 80.6% F1 score for localization alone. Visual rationales in VISPROG allow further performance gains by modifying the instructions. See Sec. 5.3 for more details on instruction tuning.

Image Editing. Tab. 4 shows the performance on the language-guided image editing task. Fig. 7 shows the wide range of manipulations possible with the current set of modules in VISPROG including face manipulations, highlight-

| Method       | Prompting strategy | Runs | Context examples per run | Accuracy |
|--------------|--------------------|------|--------------------------|----------|
| VILT-VQA     |                    | 5    | 24                       | 59.5     |
| VISPROG      | curated            | 1    | 20                       | 50.0     |
| VISPROG      | random             | 1    | 24                       | 48.2     |
| VISPROG      | voting             | 24   | 1                         | 47.8     |

Table 1. GQA testdev results. We report performance on a subset of the original GQA testdev set as described in Sec. 4.1.

| Instructions | Tagging precision | Tagging recall | Localization precision | Localization recall |
|--------------|-------------------|---------------|------------------------|---------------------|
| Original     | 69.0              | 59.1          | 63.7                   | 87.2                |
| Modified     | 77.6              | 73.9          | 75.7                   | 87.4                |

Table 2. NLVRv2 test results. VISPROG performs NLVR zero-shot i.e. without training any module on image pairs. VILT-NLVR, a VILT model finetuned on NLVRv2, serves as an upper bound.

| Instructions | Tagging precision | Tagging recall | Localization precision | Localization recall |
|--------------|-------------------|---------------|------------------------|---------------------|
| Original     |               | 59.8          |                        |                     |
| Modified     |               | 66.4          |                        |                     |

Table 3. Knowledge tagging results. The table shows performance on original instructions as well as modified instructions created after inspecting visual rationales to understand instance-specific sources of errors.

| Method       | Prompting strategy | Runs | Context examples per run | Accuracy |
|--------------|--------------------|------|--------------------------|----------|
| VILT-VQA     |                    | 5    | 24                       | 59.5     |
| VISPROG      | curated            | 1    | 20                       | 50.0     |
| VISPROG      | random             | 1    | 24                       | 48.2     |
| VISPROG      | voting             | 24   | 1                         | 47.8     |

Table 4. Image editing results. We manually evaluate each prediction for semantic correctness.
and P demonstrate the instructions to achieve better performance. To improve performance, VISPROG allows users to modify the instructions such as replacing key elements in the scene (e.g., color popping and background blur, and changing scene context by replacing key elements in the scene (e.g., desert).

5.3. Utility of Visual Rationales

**Error Analysis.** Visual rationales from VISPROG allow users to modify the instructions to improve performance. By revealing the reason for failures, VISPROG allows users to modify the original instruction to improve performance. For instance, since incorrect programs are the leading source of errors on GQA affecting 16% of samples, performance on GQA may be improved by providing more in-context examples similar to the failed questions. Performance may also be improved by upgrading models used to implement the high-error modules to more performant ones. For example, replacing the ViLT-VQA model with a better VQA model for NLVR could improve performance by up to 24% (Fig. 9).

**Instruction tuning.** To be useful, a visual rationale must ultimately allow users to improve the performance of the system on their task. For knowledge tagging and image editing tasks, we study if visual rationales can help a user modify or tune the instructions to achieve better performance. Fig. 8 shows how a localization error revealed through visual rationales enables a user to modify the instructions such that it results in a better query for the localization module. Other ways of modifying instructions include providing a better query for knowledge retrieval or a category name for the Select module to restrict search to segmented regions belonging to that category. Tables 3 and 4 show that instruction tuning results in significant gains for knowledge tagging and image editing tasks.

6. Conclusion

VISPROG proposes visual programming as a simple and effective way of bringing the reasoning capabilities of LLMs to bear on complex visual tasks. VISPROG demonstrates strong performance while generating highly interpretable visual rationales. We believe investigating new ways of incorporating user feedback to improve the performance of neuro-symbolic systems such as VISPROG is an exciting direction for building the next generation of general-purpose vision systems.

---

**Figure 7.** Qualitative results for image editing (top) and knowledge tagging tasks (bottom).

**Figure 8.** Instruction tuning using visual rationales. By revealing the reason for failures, VISPROG allows user to modify the original instruction to improve performance.

**Figure 9.** Sources of error in VISPROG.
[25] Robin Rombach, A. Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10674–10685, 2022. 2, 6

[26] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomsen, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. *ArXiv*, abs/2209.11302, 2022. 3

[27] Alane Suhr, Stephanie Zhou, Iris Zhang, Huajun Bai, and Yoav Artzi. A corpus for reasoning about natural language grounded in photographs. *ArXiv*, abs/1811.00491, 2019. 5

[28] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *ArXiv*, abs/2203.11171, 2022. 7

[29] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *ArXiv*, abs/2201.11903, 2022. 3, 7

[30] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992. 2, 3

[31] Zhengyuan Yang, Zhe Gan, Jianfeng Wang, Xiaowei Hu, Yu-mao Lu, Zicheng Liu, and Lijuan Wang. An empirical study of gpt-3 for few-shot knowledge-based vqa. In *AAAI*, 2022. 3

[32] Jiahui Yu, Yuanzhong Xu, Jing Yu Koh, Thang Luong, Gunjan Baid, Zirui Wang, Vijay Vasudevan, Alexander Ku, Yinfei Yang, Burcu Karagol Ayan, Benton C. Hutchinson, Wei Han, Zarana Parekh, Xin Li, Han Zhang, Jason Baldridge, and Yonghui Wu. Scaling autoregressive models for content-rich text-to-image generation. *ArXiv*, abs/2206.10789, 2022. 6

[33] Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, and Pete Florence. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv*, 2022. 3