ViperGPT: Visual Inference via Python Execution for Reasoning

Dídac Surís[*], Sachit Menon[*], Carl Vondrick
Columbia University
viper.cs.columbia.edu

Abstract

Answering visual queries is a complex task that requires both visual processing and reasoning. End-to-end models, the dominant approach for this task, do not explicitly differentiate between the two, limiting interpretability and generalization. Learning modular programs presents a promising alternative, but has proven challenging due to the difficulty of learning both the programs and modules simultaneously. We introduce ViperGPT, a framework that leverages code-generation models to compose vision-and-language models into subroutines to produce a result for any query. ViperGPT utilizes a provided API to access the available modules, and composes them by generating Python code that is later executed. This simple approach requires no further training, and achieves state-of-the-art results across various complex visual tasks.

1. Introduction

How many muffins can each kid in Figure 1 (top) eat for it to be fair? To answer this, we might 1) find the children and the muffins in the image, 2) count how many there are of each, and 3) reason that 'fair' implies an even split, hence divide. People find it natural to compositionally combine individual steps together to understand the visual world. Yet, the dominant approach in the field of computer vision remains end-to-end models, which do not inherently leverage this compositional reasoning.

Although the field has made large progress on individual tasks such as object recognition and depth estimation, end-to-end approaches to complex tasks must learn to implicitly perform all tasks within the forward pass of a neural network. Not only does this fail to make use of the advances in fundamental vision tasks at different steps, it does not make use of the fact that computers can perform mathematical operations (*e.g.*, division) easily without machine learning. We cannot trust neural models to generalize systematically to different numbers of muffins or children. End-to-end models also produce fundamentally uninterpretable decisions – there is no way to audit the result of each step to diagnose failure. As models grow increasingly data and compute-hungry, this approach grows increasingly untenable. We would like to perform new tasks without additional training by recombining our existing models in new ways.

In this work, we present ViperGPT[1], a framework that overcomes these bottlenecks by leveraging code generating large language models (*e.g.* GPT-3 Codex [5]) to flexibly compose vision models based on any textual query that defines the task. It creates customized programs for each query that take images or videos as argument and return the result of the query for that image or video. We show that providing Codex an API exposing various visual capabilities (*e.g.* `find`, `compute_depth`), just as one might provide an engineer, is sufficient for the creation of these programs. The model's prior training on code enables it to reason about how to use these functions and implement the relevant logic. Our results demonstrate that this simple approach delivers remarkable zero-shot performance (*i.e.* without ever training on task specific images).

Our simple approach enjoys many benefits: it is 1) *interpretable*, as all intermediate values can be inspected; 2) *logical*, explicitly using built-in Python logical and mathematical operators; 3) *flexible*, as it can easily incorporate any vision or language module; 4) *compositional*, decomposing tasks into smaller sub-tasks performed step-by-step; 5) *adaptable* to advances in the field, as improvements in any of the used modules improve the system; 6) *training-free*; and finally, 7) *general*, as it unifies all tasks into one system.

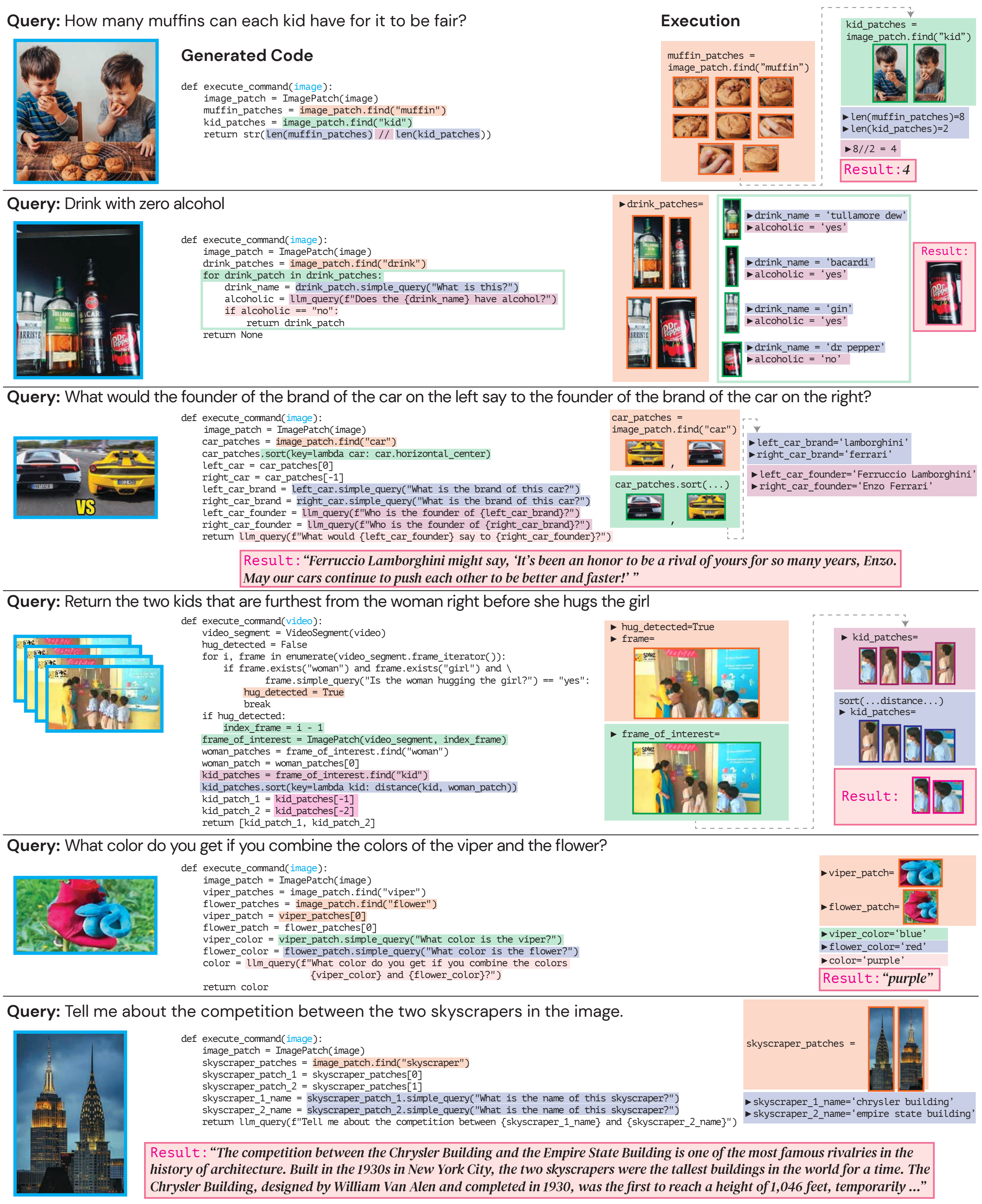
In summary: we propose a simple framework for solving complex visual queries by integrating code-generation models into vision with an API and the Python interpreter; and we achieve state-of-the-art zero-shot results across tasks in visual grounding, image question answering, and video question-answering, showing this interpretability *aids* performance rather than hindering it.

2. Related Work

We discuss related works briefly due to space limitations in this extended abstract. We take inspiration from Neural Module Networks [1, 15], who argue that complex vision tasks are fundamentally compositional and propose dividing them into atomic perceptual units. Over the course

[*]Equal contribution. Order determined via coin flip and may be listed either way.

[1]We name our method after a snake because it executes Python code.


Query: How many muffins can each kid have for it to be fair?

Generated Code

```
def execute_command(image):
    image_patch = ImagePatch(image)
    muffin_patches = image_patch.find("muffin")
    kid_patches = image_patch.find("kid")
    return str(len(muffin_patches) // len(kid_patches))
```

Execution

```
muffin_patches =
image_patch.find("muffin")

kid_patches =
image_patch.find("kid")

▶len(muffin_patches)=8
▶len(kid_patches)=2
▶8//2 = 4
Result:4
```

Query: Drink with zero alcohol

```
def execute_command(image):
    image_patch = ImagePatch(image)
    drink_patches = image_patch.find("drink")
    for drink_patch in drink_patches:
        drink_name = drink_patch.simple_query("What is this?")
        alcoholic = llm_query(f"Does the {drink_name} have alcohol?")
        if alcoholic == "no":
            return drink_patch
    return None
```

```
▶drink_patches=
▶drink_name = 'tullamore dew'
▶alcoholic = 'yes'
▶drink_name = 'bacardi'
▶alcoholic = 'yes'
▶drink_name = 'gin'
▶alcoholic = 'yes'
▶drink_name = 'dr pepper'
▶alcoholic = 'no'
Result:
```

Query: What would the founder of the brand of the car on the left say to the founder of the brand of the car on the right?

```
def execute_command(image):
    image_patch = ImagePatch(image)
    car_patches = image_patch.find("car")
    car_patches.sort(key=lambda car: car.horizontal_center)
    left_car = car_patches[0]
    right_car = car_patches[-1]
    left_car_brand = left_car.simple_query("What is the brand of this car?")
    right_car_brand = right_car.simple_query("What is the brand of this car?")
    left_car_founder = llm_query(f"Who is the founder of {left_car_brand}?")
    right_car_founder = llm_query(f"Who is the founder of {right_car_brand}?")
    return llm_query(f"What would {left_car_founder} say to {right_car_founder}?")
```

```
car_patches =
image_patch.find("car")
car_patches.sort(...)
▶left_car_brand='lamborghini'
▶right_car_brand='ferrari'
▶left_car_founder='Ferruccio Lamborghini'
▶right_car_founder='Enzo Ferrari'
```

Result: *"Ferruccio Lamborghini might say, 'It's been an honor to be a rival of yours for so many years, Enzo. May our cars continue to push each other to be better and faster!' "*

Query: Return the two kids that are furthest from the woman right before she hugs the girl

```
def execute_command(video):
    video_segment = VideoSegment(video)
    hug_detected = False
    for i, frame in enumerate(video_segment.frame_iterator()):
        if frame.exists("woman") and frame.exists("girl") and \
                frame.simple_query("Is the woman hugging the girl?") == "yes":
            hug_detected = True
            break
    if hug_detected:
        index_frame = i - 1
    frame_of_interest = ImagePatch(video_segment, index_frame)
    woman_patches = frame_of_interest.find("woman")
    woman_patch = woman_patches[0]
    kid_patches = frame_of_interest.find("kid")
    kid_patches.sort(key=lambda kid: distance(kid, woman_patch))
    kid_patch_1 = kid_patches[-1]
    kid_patch_2 = kid_patches[-2]
    return [kid_patch_1, kid_patch_2]
```

```
▶ hug_detected=True
▶ frame=
▶ frame_of_interest=
▶ kid_patches=
sort(...distance...)
▶ kid_patches=
Result:
```

Query: What color do you get if you combine the colors of the viper and the flower?

```
def execute_command(image):
    image_patch = ImagePatch(image)
    viper_patches = image_patch.find("viper")
    flower_patches = image_patch.find("flower")
    viper_patch = viper_patches[0]
    flower_patch = flower_patches[0]
    viper_color = viper_patch.simple_query("What color is the viper?")
    flower_color = flower_patch.simple_query("What color is the flower?")
    color = llm_query(f"What color do you get if you combine the colors
                       {viper_color} and {flower_color}?")
    return color
```

```
▶viper_patch=
▶flower_patch=
▶viper_color='blue'
▶flower_color='red'
▶color='purple'
Result:"purple"
```

Query: Tell me about the competition between the two skyscrapers in the image.

```
def execute_command(image):
    image_patch = ImagePatch(image)
    skyscraper_patches = image_patch.find("skyscraper")
    skyscraper_patch_1 = skyscraper_patches[0]
    skyscraper_patch_2 = skyscraper_patches[1]
    skyscraper_1_name = skyscraper_patch_1.simple_query("What is the name of this skyscraper?")
    skyscraper_2_name = skyscraper_patch_2.simple_query("What is the name of this skyscraper?")
    return llm_query(f"Tell me about the competition between {skyscraper_1_name} and {skyscraper_2_name}")
```

```
skyscraper_patches =
▶skyscraper_1_name='chrysler building'
▶skyscraper_2_name='empire state building'
```

Result: *"The competition between the Chrysler Building and the Empire State Building is one of the most famous rivalries in the history of architecture. Built in the 1930s in New York City, the two skyscrapers were the tallest buildings in the world for a time. The Chrysler Building, designed by William Van Alen and completed in 1930, was the first to reach a height of 1,046 feet, temporarily ..."*

Figure 1. **In-the-wild results.** Given a visual input and a query, ViperGPT synthesizes a program, then executes it with the Python interpreter in order to produce the final answer. This figure shows both the generated code, and the result of intermediate variables during the execution. By composing pretrained modules, ViperGPT obtains answers that are both correct and interpretable for open-world queries.

of this project, a surge of interest has resulted in a number of related manuscripts appearing on arXiv using LLMs for module integration. In NLP, they have been aimed at using external tools [19, 23], or for structured reasoning using Codex [6, 7, 9, 17, 28]. Concurrent work [11] generates a list of pseudocode instructions and interprets them as a 'visual program,' relying on in-context learning from provided examples. Unlike them, we directly generate unrestricted Python code, which is much more flexible and enables us to demonstrate more advanced emergent abilities, such as control flow and math. Crucially, using Python allows us to leverage the strong prior knowledge Codex learns by training at scale from the Internet. Additionally, we evaluate on many established benchmarks measuring visual understanding and achieve top-performing zero-shot results.

3. Method

We use notation following Johnson *et al.* [15]. Given a visual input x and a textual query q about its contents, we first synthesize a program $z = \pi(q)$ with a program generator π given the query. We then apply the execution engine $r = \phi(x, z)$ to execute the program z on the input x and produce a result r. Our framework is flexible, supporting image or videos as inputs x, questions or descriptions as queries q, and any type (*e.g.*, text or image crops) as outputs r.

3.1. Program Generation

Johnson *et al.* [15] and other work in this direction [12, 14, 31] typically implement π with a neural network that is trained with either supervised or reinforcement learning in order to estimate programs from queries. However, these approaches have largely been unable to scale to in-the-wild settings because either a) the supervision in the form of programs cannot be collected at scale or b) the optimization required for finding the computational graph is prohibitive.

In our approach, we instead capitalize on LLMs for code generation in order to instantiate the program generator π that composes vision and language modules together. LLMs take as input a tokenized code sequence ("prompt") and autoregressively predict subsequent tokens. We use Codex [5], which has shown remarkable success on code generation tasks. Since we replace the optimization of π with an LLM, our approach obviates the need for task-specific training for program generation. Using Codex as the program generator and generating code directly in Python allows us to draw on training at scale on the Internet, where Python code is abundant.

To leverage LLMs in this way, we need to define a prompt that will sample programs z that compose and call these modules as needed. Our prompt consists of an application programming interface (API), detailed in the following section, which we provide to the LLM as part of its input context. The final input to the LLM is a sequence of code text consisting of the API specification followed by the query for the sample under consideration. The expected output is a Python function definition as a string, which we then compile and execute.

3.2. Modules and Their API

The API we provide defines two global classes `ImagePatch` and `VideoSegment`, which represent an image patch and a video segment respectively. Each module is implemented as a class method, which internally calls a pretrained model to compute the result. The API specifies the input and output types for each method it defines, as well as docstrings to explain the purpose of these functions in natural language. Like most APIs, it additionally provides examples that show how to use these classes and their functions, specified in the form of query-code pairs similarly to in-context learning [3, 25].

3.3. Program Execution

At execution time, the generated program z accepts an image or video as input and outputs a result r corresponding to the query provided to the LLM. To execute this program, previous work (*e.g.*, [15]) learns all neural modules together simultaneously end-to-end, which fails to enable systematic generalization [2] and results in modules that are not *faithful* to their intended tasks [24], compromising the interpretability of the model.

The program is run with the Python interpreter; as such, *its execution is a simple Python call.* This means it can leverage all built-in Python functions like `sort`; control flow tools like `for` or `if/else`; and modules such as `datetime` or `math`. The Python interpreter enables logical operations while the pretrained models enable perceptual ones. Our approach guarantees faithfulness by construction. Notably, this does not require a custom interpreter, unlike prior approaches [11, 23] Another advantage of a fully Pythonic implementation is compatibility with a wide range of existing tools, such as PyTorch JIT [20].

4. Evaluation

`ViperGPT` is applicable to any tasks that query visual inputs with text. Unlike other work using large language models for vision tasks, the return values of our programs can be of arbitrary types, such as text, multiple choice selections, or image regions. We select four different evaluation settings to showcase the model's diverse capabilities in varied contexts without additional training. We evaluate on tasks we consider to build on each other: 1) visual grounding, 2) compositional image question answering, 3) external knowledge-dependent image question answering, and 4) video causal and temporal reasoning.

Table 1. Comparison of our method with state-of-the-art models on various tasks.

	RefCOCO Results			GQA Results		OK-VQA Results		NExT-QA Results			
		RefCOCO	RefCOCO+		Accuracy		Accuracy		Hard Split - T	Hard Split - C	Full Set
Sup.	MDETR [27]	90.4	85.5	LGCN [13]	55.8	TRiG [8]	50.5	ATP [4]	45.3	43.3	54.3
Sup.	OFA [27]	94.0	91.7	LXMERT [26]	60.0	KAT [10]	54.4	HiTeA [30]	48.6	47.8	63.1
ZS	OWL-ViT	30.3	29.4			PICa	43.3				
ZS	GLIP	55.0	52.2	FewVLM	29.3	BLIP-2	45.9				
ZS	ReCLIP	58.6	60.5	BLIP-2	44.7	Flamingo	50.6				
ZS	ViperGPT (ours)	**72.0**	**67.0**	ViperGPT (ours)	**48.1**	ViperGPT (ours)	51.9	ViperGPT (ours)	**49.8**	**56.4**	60.0

4.1. Visual Grounding

Visual grounding is the task of identifying the bounding box in an image that best matches a text query, requiring spatial reasoning and visual attribute understanding.

We provide ViperGPT with APIs for the following modules (pretrained models in parentheses): **find** (GLIP [16]) takes an image and a noun phrase, returning a list of image patches containing the noun phrase; **exists** (GLIP [16]) takes an image and a noun phrase, returning a boolean indicating the presence of the noun phrase in the image; **verify_property** (X-VLM [32]) takes an image, a noun phrase, and an attribute, returning a boolean indicating the presence of the property in the image; **best_image_match** (X-VLM [32]) takes a list of image patches and a noun phrase, returning the best-matching image patch; **best_text_match** takes a list of noun phrases and an image, returning the best-matching noun phrase (not necessary for visual grounding, but included for simplicity), implemented using an image-text similarity model like CLIP [21]; and **compute_depth** (MiDaS [22]) computes the median depth of an image patch. We also define **distance**, computing pixel-distance between two patches using built-in Python tools.

We evaluate using RefCOCO and RefCOCO+ datasets, with the former allowing spatial relations and the latter not, offering insights into ViperGPT's capabilities. We compare ViperGPT against end-to-end methods, outperforming other zero-shot methods on both datasets (see Table 1).

4.2. Compositional Image Question Answering

We evaluate ViperGPT on image question answering, focusing on compositional tasks using the GQA dataset. Providing intermediate reasoning is more interpretable and human-aligned; as our final result is constructed directly from the intermediate values, they provide a fully faithful interpretation of how the model came to its answer..

For GQA, we incorporate the module **simple_query** (BLIP-2 [16]), which handles basic queries that are not further decomposable, such as "What animal is this?" We also add the aforementioned **best_text_match**. This leads us to the best accuracy on GQA among zero-shot models (Table 1).

4.3. External Knowledge-dependent Image Question Answering

Integrating external knowledge about the world is crucial for answering many image-related questions. We equip ViperGPT with a natural language module to query external knowledge bases, **llm_query** (GPT-3 [3]), enabling it to combine knowledge with visual reasoning.

We assess ViperGPT on the OK-VQA dataset [18], designed to evaluate models' abilities to answer questions requiring extrinsic knowledge. This dataset often involves multi-step reasoning to produce correct answers. By using a form of chain-of-thought reasoning [29], ViperGPT uses perceived information and the external knowledge module to generate accurate responses.

ViperGPT outperforms all zero-shot methods and surpasses the best previous model using publicly available resources by 6% (Table 1), a significant margin for this dataset.

4.4. Video Causal/Temporal Reasoning

We also evaluate how ViperGPT extends to videos and queries that require causal and temporal reasoning. To explore this, we use the NExT-QA dataset, designed to evaluate video models ability to perform this type of reasoning. We evaluate using the NExT-QA multiple choice version.

We provide an additional module **select_answer** (GPT-3 [3]), which, given textual information about a scene and a list of possible answers, returns the answer that best fits the information. Other than that, the only additional content given in the API is the definition of the class `VideoSegment`, that contains the video bytestream as well as the start and end timestamps of the video segment that it represents. It also defines an iterator over the frames, which returns an `ImagePatch` object representing every frame.

Despite only having image perception modules, ViperGPT demonstrates emergent causal and temporal reasoning in videos, generating programs that identify relevant frames for queries and producing accurate results. Its accuracy is on par with, or even surpasses, supervised models for temporal (T) and causal (C) reasoning in Table 1, and incorporating video models could potentially enhance performance further.

References

[1] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 1

[2] Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron Courville. Systematic Generalization: What Is Required and Can It Be Learned?, Apr. 2019. arXiv:1811.12889 [cs]. 3

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. *arXiv:2005.14165 [cs]*, July 2020. arXiv: 2005.14165. 3, 4

[4] Shyamal Buch, Cristóbal Eyzaguirre, Adrien Gaidon, Jiajun Wu, Li Fei-Fei, and Juan Carlos Niebles. Revisiting the" video" in video-language understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2917–2927, 2022. 4

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, David W. Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William H. Guss, Alex Nichol, Igor Babuschkin, S. Arun Balaji, Shantanu Jain, Andrew Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew M. Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374, 2021. 1, 3

[6] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022. 3

[7] Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. Binding language models in symbolic languages. *International Conference on Learning Representations (ICLR)*, 2023. 3

[8] Feng Gao, Qing Ping, Govind Thattai, Aishwarya Reganti, Ying Nian Wu, and Prem Natarajan. Transform-retrieve-generate: Natural language-centric outside-knowledge visual question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5067–5077, 2022. 4

[9] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*, 2022. 3

[10] Liangke Gui, Borui Wang, Qiuyuan Huang, Alexander Hauptmann, Yonatan Bisk, and Jianfeng Gao. KAT: A knowledge augmented transformer for vision-and-language. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 956–968, Seattle, United States, July 2022. Association for Computational Linguistics. 4

[11] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. *arXiv preprint arXiv:2211.11559*, 2022. 3

[12] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to Reason: End-to-End Module Networks for Visual Question Answering. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 804–813, Oct. 2017. Conference Name: 2017 IEEE International Conference on Computer Vision (ICCV) ISBN: 9781538610329 Place: Venice Publisher: IEEE. 3

[13] Ronghang Hu, Anna Rohrbach, Trevor Darrell, and Kate Saenko. Language-conditioned graph networks for relational reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10294–10303, 2019. 4

[14] Drew A. Hudson and Christopher D. Manning. Compositional Attention Networks for Machine Reasoning. *ArXiv*, 2018. 3

[15] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. Inferring and Executing Programs for Visual Reasoning. pages 2989–2998, 2017. 1, 3

[16] Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, et al. Grounded language-image pre-training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10965–10975, 2022. 4

[17] Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. Language models of code are few-shot commonsense learners. *arXiv preprint arXiv:2210.07128*, 2022. 3

[18] Kenneth Marino, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. OK-VQA: A Visual Question Answering Benchmark Requiring External Knowledge. May 2019. 4

[19] Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255*, 2022. 3

[20] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An

imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 3

[21] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 4

[22] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(3), 2022. 4

[23] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023. 3

[24] Sanjay Subramanian, Ben Bogin, Nitish Gupta, Tomer Wolfson, Sameer Singh, Jonathan Berant, and Matt Gardner. Obtaining Faithful Interpretations from Compositional Neural Networks, Sept. 2020. arXiv:2005.00724 [cs]. 3

[25] Dídac Surís, Dave Epstein, Heng Ji, Shih-Fu Chang, and Carl. Vondrick. Learning to learn words from visual scenes. *European Conference on Computer Vision (ECCV)*, 2020. 3

[26] Hao Tan and Mohit Bansal. Lxmert: Learning cross-modality encoder representations from transformers. *arXiv preprint arXiv:1908.07490*, 2019. 4

[27] Peng Wang, An Yang, Rui Men, Junyang Lin, Shuai Bai, Zhikang Li, Jianxin Ma, Chang Zhou, Jingren Zhou, and Hongxia Yang. Ofa: Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework. *CoRR*, abs/2202.03052, 2022. 4

[28] Xingyao Wang, Sha Li, and Heng Ji. Code4struct: Code generation for few-shot structured prediction from natural language. *arXiv preprint arXiv:2210.12810*, 2022. 3

[29] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain of Thought Prompting Elicits Reasoning in Large Language Models, Oct. 2022. arXiv:2201.11903 [cs]. 4

[30] Qinghao Ye, Guohai Xu, Ming Yan, Haiyang Xu, Qi Qian, Ji Zhang, and Fei Huang. Hitea: Hierarchical temporal-aware video-language pre-training. *arXiv preprint arXiv:2212.14546*, 2022. 4

[31] Kexin Yi, Jiajun Wu, Chuang Gan, A. Torralba, Pushmeet Kohli, and J. Tenenbaum. Neural-Symbolic VQA: Disentangling Reasoning from Vision and Language Understanding. *ArXiv*, 2018. 3

[32] Yan Zeng, Xinsong Zhang, and Hang Li. Multi-grained vision language pre-training: Aligning texts with visual concepts. *arXiv preprint arXiv:2111.08276*, 2021. 4