Tool Documentation Enables Zero-Shot Tool-Usage with Large Language Models

Cheng-Yu Hsieh[1†], **Si-An Chen**[2†], **Chun-Liang Li**[3], **Yasuhisa Fujii**[4], **Alexander Ratner**[1], **Chen-Yu Lee**[3], **Ranjay Krishna**[1*], **Tomas Pfister**[3*]
[1]University of Washington, [2]National Taiwan University,
[3]Google Cloud AI Research, [4]Google Research
cydhsieh@cs.washington.edu

Abstract

Today, large language models (LLMs) are taught to use new tools by providing a few demonstrations of the tool's usage. Unfortunately, demonstrations are hard to acquire, and can result in undesirable biased usage if the wrong demonstration is chosen. Even in the rare scenario that demonstrations are readily available, there is no principled selection protocol to determine how many and which ones to provide. As tasks grow more complex, the selection search grows combinatorially and invariably becomes intractable. Our work provides an alternative to **demonstrations**: tool **documentation**. We advocate the use of tool documentation—descriptions for the individual tool usage—over demonstrations. We substantiate our claim through three main empirical findings on 6 tasks across both vision and language modalities. First, on existing benchmarks, zero-shot prompts with only tool documentation are sufficient for eliciting proper tool usage, achieving performance on par with few-shot prompts. Second, on a newly collected realistic tool-use dataset with hundreds of available tool APIs, we show that tool documentation is significantly more valuable than demonstrations, with zero-shot documentation significantly outperforming few-shot without documentation. Third, we highlight the benefits of tool documentations by tackling image generation and video tracking using just-released unseen state-of-the-art models as tools. Finally, we highlight the possibility of using tool documentation to automatically enable new applications: by using nothing more than the documentation of GroundingDino, Stable Diffusion, XMem, and SAM, LLMs can *re-invent* the functionalities of the just-released Grounded-SAM [23] and Track Anything [70] models.

1 Introduction

Today, large language models (LLMs) summon the imagery of a craftsman: when asked to solve a complex task, they decompose the task into simpler sub-tasks and assemble the best possible tools to tackle each sub-task [51, 72]. For example, consider the complex task of question answering given the image in Figure 1. To answer "whether the two magnets will attract or repel each other", the LLM needs the following: it needs to identify the positions of the magnets in the image, extract general knowledge explaining that "opposite (same) poles attract (repel)". Just like a competent craftsman who knows what their tools are capable of, an LLM with such knowledge of its tools will be able to invoke one tool (e.g. its Text Detector) to identify the north and south poles and a second tool (e.g. Knowledge Retriever) to extract pertinent background knowledge about magnetic forces. But how does an LLM know which tool is capable of what?

[†]Work done as student researchers at Google Cloud AI Research.
[*]The authors contributed equally to this work.

Preprint. Under review.

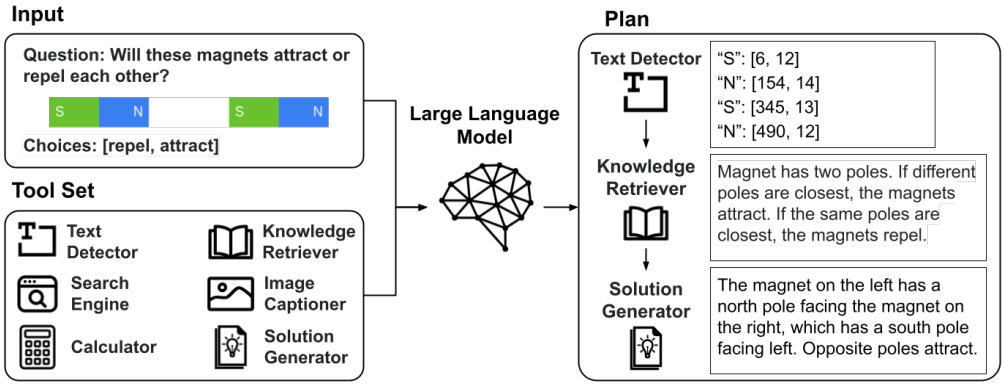

Figure 1: Example workflow of tool-using with LLMs to solve a multi-modal question answering task. Given the input question with an image, the LLM selects appropriate tools from the tool set and generates an execution plan to answer the question correctly. Here, the LLMs outlines a plan to first use Text Detector to understand the positioning of the magnets in the image, then leverage Knowledge Retriever to obtain relevant background knowledge about magnets, then finally generate the solution based on the previous steps.

Currently, LLM tool-usage provides LLMs with few-shot demonstrations (demos) of what its tools can do, hoping that these demos will help generalize the model's behavior to newer complex tasks. This process has been rather successful so far. These few-shot demos contain one or several exemplars of <input, output> mappings [68] on given instructions and their corresponding tool-use plans (illustrated in Figure 2). LLMs are expected to find patterns within these demos and generalize them for new tasks. On textual tasks, LLMs have presented with demos of calculators [15, 47, 56], Python interpreters [13, 18] and search engines [62, 43, 50, 56, 40] can perform logical and arithmetic operations to obtain more accurate and factual knowledge. On visual tasks, LLMs with demos of pretrained vision models can do complex visual reasoning [37, 40, 57, 16, 73], can generate and even edit images [19, 9]. On embodied robotic tasks, LLMs can similarly be used to reason and plan [75, 21, 1, 17].

We argue that this reliance on demos in tool using is unnecessary in some cases, and might be even limiting. In fact, recent work finds that LLMs tend to be sensitive to demos [81], and carefully selecting demos is needed to avoid biasing or overfitting to a particular usage [12]. This leads to the follow-up question: how do we choose which few-shot demos to use? There are no known principled approaches to select demos without human intervention or to even efficiently enable humans to choose or create them. To make the matter worse, when we scale up the number of tools that LLMs have access to, this few-shot selection process becomes combinatorially intractable. Just as a craftsman doesn't need to see a new tool being demonstrated and can instead discern their capabilities from reading a user manual for the tool, we seek to enable LLMs to learn how to use tools without seeing any demos.

Our work provides an alternative to **demonstrations**: tool **documentation** (doc). Similar to the metaphor of a manual indicating an physical tool's capabilities, a software tool's docs outline what the tool can and cannot be used for and how to invoke it. Docs provide relatively neutral instruction about the tools' functionalities and how individual tools should be used (illustrated in Figure 2), and they are usually conveniently available through the creation of the tools organically. Intuitively, just as the craftman leans to use a new tool by reading the manual, we provide LLMs with `README` files when encountering a new tool/repository. With docs, an LLM may not necessarily need demos to use a new tool.

Distinct from existing work that rely mostly on few-shot demos for tool-learning, in this work, we study whether LLMs can instead solely rely on docs to use tools. We study the tool-learning performances of LLMs as we include or exclude docs, and vary the number of demos from few-shot down to zero-shot. We conduct the experiments on 6 tasks across vision and text modalities. Our experiments show that:

- Surprisingly, when provided with tool docs, LLMs' zero-shot tool-using performance is on par or even better than their few-shot counterparts, showing that including docs is an effective way to sidestep the few-shot demos needed.

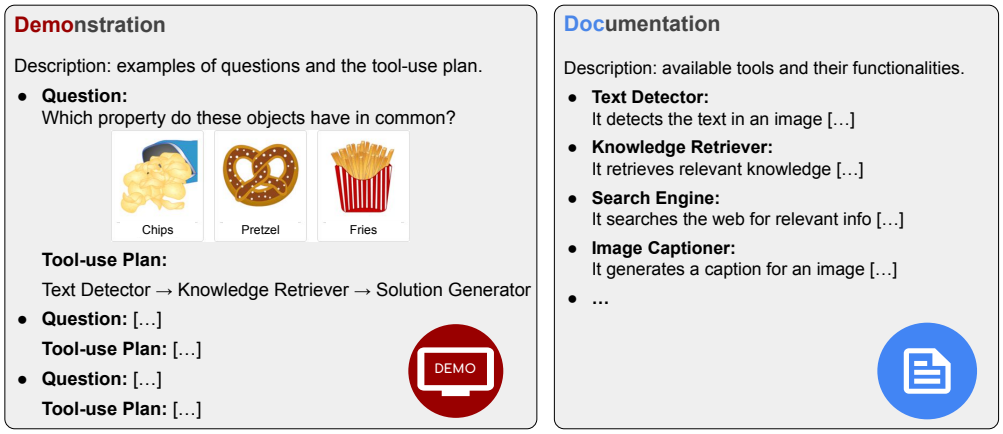

Figure 2: Two types of knowledge for prompting LLMs for tool-use: Demonstrations (demos) and Documentations (docs). Demos consist of <input, output> pairs on input instructions and their corresponding output tool-use plans. They require manual efforts for careful curation on every new task, and the model performance can be sensitive to which demos are used [81, 12]. Many demos may also be necessary for good coverage when the number of tools scales up. On the other hand, docs provide descriptions for the tool functionality, and are usually organically available for tools.

- Building on the above finding, we relax the few-shot demo constraint, and show that we can efficiently scale up to a significantly larger tool set, on a newly collected API usage dataset, by simply providing the LLMs with docs.
- We show how to seamlessly add new tools along with their docs to a tool set for LLMs to solve unseen tasks on image editing and video tracking, all without any further demos in a plug-and-play manner.
- Finally, with unseen tools developed recently as building blocks, we showcase LLMs are capable of *re-inventing* popular yet even more recent works Grounded-SAM [23] and Track Anything [70], which suggests a potential from zero-shot tool usage to automatic knowledge discovery.

2 Related work

LLMs with retrieval augmentation and tools. In spite of the remarkable achievements demonstrated by LLMs, the performance can be further boosted with external tool usages to be more accurate, efficient or versatile for wider applications. The authors in [51] detailed the cognitive origins, the paradigm shift of foundation models, and the complementary roles of tools and models to LLMs. The example tool usage starts from knowledge retrieval [6, 20, 33, 74, 77] and expands to search engine [43, 31, 32, 62, 58, 46, 40], QA system [56], calculator [15, 47, 56], the Python interpreter [18, 13, 65, 24, 46, 16], simulation engines [37], machine learning models [57, 73, 69, 40, 16], or even tools created by LLMs [11]. Pioneer works of LLMs with tools often rely on human supervision [62, 31] or additional self-supervised learning techniques [56], which pose challenges for practical plug-and-play usage. Recent advancements eliminate additional training by using example demos in the prompt [19, 75, 73, 57, 40, 46]. Our work further simplifies prompt design by only leveraging documentation for individual tools, while maintaining competitive performance.

Planning with LLMs. Language models are proven to have potential to conduct planning for solving complex tasks or decompose the complex tasks into sub-problems when prompted properly. [21, 22] retrieve demos at test-time with large knowledge space coverage to generate admissible actions. [28] relies on pre-designed demos for task decomposition. Similarly, recent works of tool using with LLMs leverage the example demonstrations of solving examples tasks with a planning of tools [13, 19, 75, 73, 57, 40, 46]. However, crafting demos of interactions between tools may be challenging in practice when the number of tools surges. Concurrent work [48, 52, 71] tackles the challenge by using strong LLMs such as GPT-4 [45] to create large instruction-following datasets that cover diverse instructions and corresponding tool-use plans, typically through mechanisms like

self-instruct [66]. The resultant datasets can then be used to finetune and equip other LLMs (e.g., LLaMA [63] and OPT [79]) the ability to use a large collection of tools for unseen instructions. On the other hand, our work showcases the potential for LLMs to utilize any unseen new tools by reading their tool docs.

Demonstration and Documentation. Learning from demonstration is popular in reinforcement learning [49, 4, 44, 55]. [8] propose the in-context learning algorithm for efficient and effective downstream task adaptations through showing example demonstrations. Inspired by the success, most of existing LLM tool-using works rely on few-shot demonstration [13, 19, 75, 73, 57, 40, 46]. However, [12] show that having more example demonstration might counter-intuitively degrade performance, and a careful selection might be needed. [35] proposes a retrieval method for demo selection, which implicitly requires a larger set of examples to be selected. Using documentation to improve algorithms is relatively under-explored. [7, 82] propose document reading algorithms for specific games. [83] introduced DocPrompting, which employs a trained retriever on the given training data to boost code generation by retrieving relevant documents. In this work, we take a step towards exploring the zero-shot tool planning in LLMs solely with the aid of documentation, and investigate a wide range of diverse tasks from language to vision domains. While [64, 42] showcase pure zero-shot planning capability of LLMs, they do not study either the tool usage or the unseen scenarios to the language models. ViperGPT [16] is a concurrent work, which focuses on visual programming in Python and uses function implementations and specifications as documentation. Lastly, while AutoGPT [3] provides several demos that showcase the LLM's capability of tool using through documentation reading, our study focuses on a systematic exploration ranging from real-world use cases to academic benchmarks.

3 Experimental setup

3.1 General workflow

We follow the general framework of tool-using with LLMs in [51], which encompasses many of the recent works [75, 27, 19, 57, 73, 69, 40]. Specifically, given a natural language instruction, an *LLM planner* generates a *program* to be sequentially executed where each step of the program may rely on using tools selected from a *tool set*. After the program is generated, it is then executed by an *environment* which finally returns the execution results. Here, the *program* extends beyond conventional coding practice [76, 53, 25] and is more closely associated with automata theory [59]: a set of instructions of automations (e.g. tools in our case). Therefore, the tool set can be libraries with specific programming languages (e.g. Python), or general computation with properly defined input-output, such as trained models, API calls, and beyond.

3.2 Tool-use prompting methods

As discussed in Section 1, two main types of information are considered in prompting LLMs for tool-using plans: demonstrations (demos) and documentations (docs). Demos showcase how tool *interactions* can accomplish *specific* tasks, while docs describe individual tool functionalities without task-specific ties as shown in Figure 2. In the experiment, we explore combinations of including/excluding docs and demos in prompts, as well as varying numbers of demos.

3.3 Evaluation tasks

We conduct our experiments on 6 tasks across multiple modalities with a variety of tool sets. We describe the setup and the tool sets for each task below. Except for specific cases where it is explicitly specified, the LLM planner is ChatGPT (`gpt-3.5-turbo`).

Multi-modal question answering on ScienceQA. ScienceQA [39] consists of multi-modal multiple-choice science questions that requires language and visual understanding as well as domain-specific knowledge to answer correctly. On ScienceQA, we follow the setup used in Chameleon [40] and employ the same tool set with 7 tools, such as the search engine and the image text detector.

Tabular math reasoning on TabMWP. TabMWP [41] is a math reasoning dataset with various forms of tables. It requires a model to understand structured or domain-specific tables, and utilize the

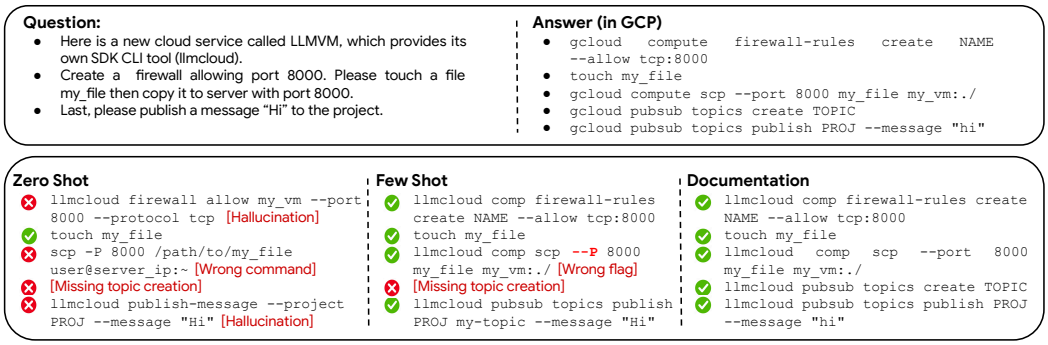

Figure 3: The new LLM Cloud Platform command-line toolkit, which is an unseen toolset to existing LLMs based on real-world Google Cloud command-line tools through renaming.

information to answer corresponding math questions. On TabMWP, we also follow Chameleon [40] with the same tool set with 9 tools, such as program generator and column lookup.

Multi-modal reasoning on NLVRv2. NLVRv2 [60] requires the model to verify whether a statement is true on a pair of images, requiring compositional understanding of both texts and images. On NLVRv2, we follow the setup used in Visual Programming (VisProg) [19] with 20 vision modules (tools) for image understanding and manipulation. Since VisProg only relies on few-shot demonstrations and does not utilize documentations for the modules. We generate the documentation for each module by including descriptions on the functionality of the module and the function signature. We provide the full documentations we use for each module in the appendix.

Unseen API usage on a newly collected dataset. Existing benchmarks used in literature come with a limited set of tools. To explore real-world use cases involving a large number of tools, we collect a new benchmark called the *LLM Cloud CLI* that consists of 200 commands representing the functionalities of the Google Cloud Platform (GCP) command-line interface (CLI). Each command in our CLI is renamed from its corresponding GCP command, preserving the semantics and logic of the original tools, while being unseen to the language models. For instance, the command `gcloud compute create NAME`, responsible for creating a virtual machine, is renamed to be `llmvm compute make NAME`. The renaming conventions also allow us to utilize authentic GCP examples as few-shot demos and leverage the corresponding GCP documentation. The benchmark comprises 50 questions, each focused on creating and configuring specific cloud services using command-line tools. Each question requires at least two commands to complete the task. We show an example in Figure 3, and include more in appendix.

Due to the length constraints of the LLM we use, we cannot fit documentation of 200 tools in a single prompt. Therefore, we employ a simple TF-IDF search using the questions as queries to retrieve the most relevant documentations and truncate them to fit within the prompt length. More details can be found in the appendix.

Image editing with natural language. We consider image editing as a form of qualitative evaluation. This process calls for the model to plan and use different vision modules to handle complex natural language instructions. For instance, to execute an instruction like "replace the red bus with a green bicycle", the model must localize the red bus, generate its segmentation mask, and then inpaint the masked area. We use the tool sets from VisProg. Unlike VisProg, which depends on few-shot demonstrations, our model only looks at the module documentation. We further include the recently released image understanding works, Segment Anything (SAM) [30] and Grouding DINO [38] to expand the tool set to test the zero-shot capability on the new and unseen tools in a plug-and-play fashion.

Video tracking. Video tracking is also utilized in this study as a qualitative evaluation. This task aims to acquire the masks of a tracked object in each frame of a video, necessitating the deployment of processes such as object localization, segmentation, and tracking. In addition to SAM and Groudning DINO, we incorporate the documentation of an unseen object tracking module, Xmen [14] into the VisProg framework with the aim to showcase the model's ability to adapt and employ new tools without the need for explicit demonstrations again on a different task.

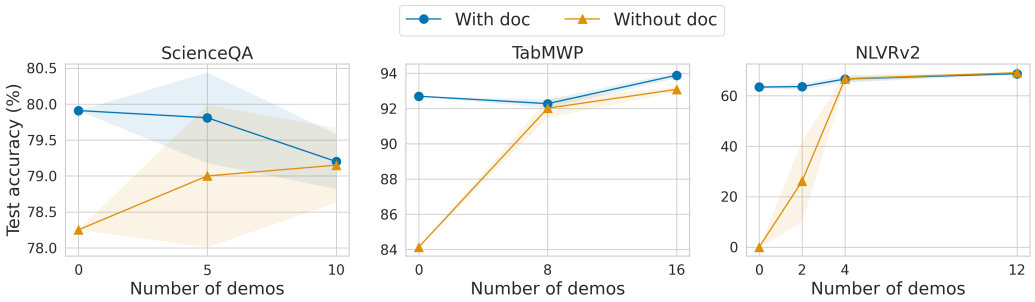

Figure 4: Tool-using performance with `gpt-3.5-turbo` on different benchmarks, which covers from langauge to vision modalities. We report results with and without documentation (doc) and demonstations (demo), and their combinations. Clearly, with documentation only (upper-left blue dot) shows competitive performance across all datasets.

4 Empirical findings

We showcase the importance of tool documentation in three-fold: First, we show that tool documentations reduces the need of demonstrations (Section 4.1). Second, based on the finding, we further show that relying on documentation rather than demonstrations provides a more scalable solution to equip LLMs with a large number of available tools (Section 4.2). Finally, we show that with tool documentations alone, LLMs are able to comprehend and utilize most recent vision models to accomplish impressive results on image editing and video tracking tasks, on which existing results are achieved either with human-crafted demos or predefined procedures (Section 4.3).

4.1 Documentations sidestep the need for demonstrations

In this section, we show how tool documentations reduce the need of demonstrations. We present the findings on three datasets: ScienceQA, TabMWP, and NLVRv2. We evaluate the model performance, with and without tool documentations, across varying number of demonstrations (demo) on each dataset.

In Figure 4, we see that when provided with tool docs, the model is able to maintain stable performance as we strip away the number of demos used. In fact, without using any demos (i.e., 0-shot), the model is able to achieve on par performances to using 16-shot on TabMWP, and using 12-shot on NLVRv2. On ScienceQA, the model can even achieve better performance solely with docs compared to additionally using 10-shot demos. On the other hand, without tool docs, the model performance is very sensitive to the number of demos used. As we decrease the number of demos, we see significant performance drop on all three datasets. This highlights the importance of tool docs and shows that it provides an effective way to reduce the reliance on demos. In Table 1, when compared to existing baseline methods, we also see that with doc, even 0-shot can perform very competitively.

By sidestepping the need for demos, we are able to alleviate the efforts needed to carefully curate these demos. For example, aligned with recent studies [81, 12], we observe in Figure 4 that the model performance is sensitive to which demos are used, shown by the large performance variances under 5-shot on ScienceQA and 2-shot on NLVRv2.

4.2 Documentations enable efficient scaling on tool-using

The findings in Section 4.1 show that one can in fact reduce the reliance on few-shot demos with tool docs. By relaxing this constraint, we study whether tool docs enables a more scalable way to equip LLMs with a large number of tools, wherein few-shot demos can specifically fall short on covering limited tool-use cases. We present our findings in this section on the newly collected LLM Cloud CLI dataset with 200 available tools.

Qualitative walk-through result. Figure 3 serves as a qualitative example illustrating the limitations of the LLMs with different information. As expected, zero-shot LLM successfully identifies and responds to the `touch` command, which is familiar and well-known. However, when faced with the

Table 1: Comparisons to existing baseline methods on different benchmarks. We follow [40, 19] to select the beasline methods for each benchmark task. We see that 0-shot with doc performs competitively, outperforming CoT and PoT on ScienceQA and TabMWP. On NLVRv2, ViLT-NLVR is finetuned on the dataset, while the LLM performs in a zero-shot fashion.

Benchmark	Methods		
	CoT [67]	without doc (0-shot)	with doc (0-shot)
ScienceQA	78.54	78.25	**79.91**
	PoT [13]	without doc (0-shot)	with doc (0-shot)
TabMWP	89.28	84.13	**92.69**
	ViLT-NLVR [29]	without doc (0-shot)	with doc (0-shot)
NLVRv2	**76.30**	0.00	63.40

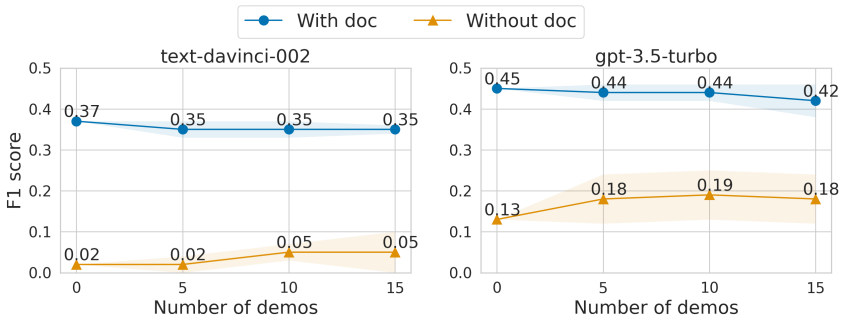

Figure 5: Command planning of LLM Cloud Platform CLI with and without documentation (doc) and demonstations (demo), and their combinations. Few-shot demonstration without documentation results in unsatisfactory performance due to low coverage of large number of tools, while reading documentation significantly boosts the performance.

unseen LLM-Cloud command lines, the zero-shot LLM fails to generate accurate responses involving these unfamiliar tools due to its lack of knowledge regarding their syntax and usage.

While few-shot demonstrations have the potential to enhance model performance, it is important to acknowledge that the coverage of these demonstrations is limited due to the vast number of command-line tools. Consequently, certain commands or flags may not be adequately covered. In Figure 3, although we observe data copying is commonly appeared the few-shot examples, however, the model encounters difficulties in correctly configuring the less common flag `--port`, instead hallucinating the use of `-P` based on familiarity with the `scp -P` command in Linux.

Conversely, in the same example illustrated in Figure 3, by solely utilizing the provided documentation, the language models not only successfully discern the steps required for utilizing tools (such as a hidden step of creating a `topic` before sending messages), but also possess the ability to accurately configure flags (e.g., `--port`) by leveraging information extracted from the documentation.

Quantitative comparisons. We calculate the command-line level F1 score of each example and report the average F1 across 50 examples. Figure 5 showcases the performance of various LLMs in the zero-shot setting, where they have no prior exposure to the LLM-Cloud command-line tools we create. As anticipated, all zero-shot LLMs demonstrate low F1 scores. Zero-shot `text-davinci-002` achieves an F1 score of 0.02, while the `gpt-3.5-turbo` model achieves a slightly higher score of 0.13. The improved performance of the `gpt-3.5-turbo` model can be attributed to better handling of common Linux commands, such as `touch`. As mentioned in quantitative comparison, few-shot demos improve upon zero-shot, but still fail on uncovered commands or flags in the demo. Therefore, the best few-shot demo in `text-davinci-002` and `gpt-3.5-turbo` are only with 0.05 and 0.19 F1 scores respectively. On the other hand, LLM with documentation boosts the performance by a large margin to be 0.37 in `text-davinci-002` and 0.45 in `gpt-3.5-turbo` .

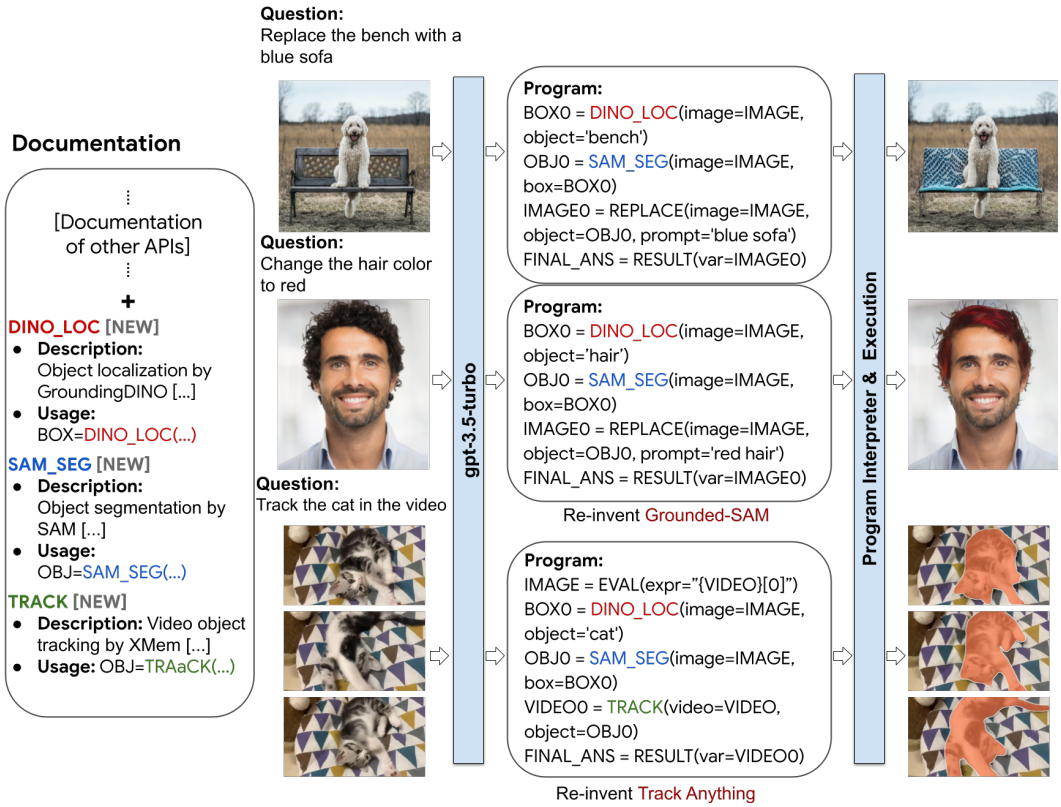

Figure 6: Plug-and-play new vision tools without demonstration. We add GroundingDINO [38], Segment Anything (SAM) [30], XMem [14] as new tools for VisProg. Solely with the documentations of the new tools, the LLM is able to automatically "re-invent" recent Grounded-SAM [23] and Track Anything [70] without knowing these derivatives, taking a further step toward automatic knowledge discovery.

We further compare the performance of the documentation reading with that of the documentation supplemented with few-shot demonstrations. In the case of `text-davinci-002` , with documentation only, we achieves an F1 score of 0.37. Conversely, the documentation augmented with different shots yields an average F1 score of 0.35. Similarly, in the `gpt-3.5-turbo` experiment, the performance with different shot demonstrations (0.44, 0.44, 0.42) are consistently lower than the documentation-only performance (0.45).

These results highlight two observations. First, the performance of the model is highly sensitive to the selection of few-shot demonstrations. The observation aligns the finding in [12] that more few-shot demos might be redundant and even degrade performance due to spurious correlations. It emphasizes the importance of careful selection and design, which may involve more human effort. Second, the zero-shot documentation reading baseline exhibits remarkable robustness and delivers competitive performance across both examples. This highlights the potential value and reliability of relying solely on the documentation, which is usually easy to get in many packages and tools.

4.3 Plug-and-play with new image and video tools

In this section, we validate that one can equip LLMs with unseen tools to solve novel tasks solely with tool docs, and without any further demos. We present our results on image editing and video tracking tasks. We show that LLMs can effectively re-invent existing human-programmed image editing and video tracking pipelines, backed by state-of-the-art vision models to achieve impressive results.

Recent advancements in vision models, including GroundingDINO [38], an advanced open-set object detector; Segment Anything (SAM) [30], a cutting-edge image segmentation tool; and XMem [14], a

state-of-the-art video object segmentation tool, accompany the progress of language models. These breakthroughs, emerging in the past year, serve as additional tools that are yet unfamiliar to our LLM (`gpt-3.5-turbo`). By expanding VisProg to include these new tools, we embark on the intriguing exploration of whether LLMs can effortlessly comprehend the documentation associated with these new models, and combine these tools in a plug-and-play manner, enabling a wide range of applications.

In Figure 6, when performing an image editing request "replace the bench with a blue sofa", the LLM generates a VisProg program that harnesses the power of GroundingDINO and SAM from the expanded tool set to segment the bench, and apply the stable diffusion [54] for synthesizing the sofa. This program *re-invents the wheel* by replicating the behavior of recent popular project, Grounded-SAM [23] without prior knowledge of this repository. Similarly, when tasked with video tracking "track the cat in the video", the generated VisProg program by the LLM incorporates GroundingDINO together SAM for first frame segmentation as the initialization for XMem to do video tracking. It again re-invents the results obtained in the contemporary work, Track Anything [70]. We note that TaskMatrix [69] also has an updated approach with Grounded-SAM. However, they pre-program the entire Grounded-SAM editing pipeline as an image editing function, allowing the LLM to control it rather than enabling the LLM to generate the editing program using the building tools alone as we present here.

By successfully re-inventing the functionalities of Grounded-SAM and Track Anything without prior knowledge, solely relying on the available building blocks, the LLM demonstrates not only its capacity to effortlessly comprehend and combine new tools with documentation only but also highlights its potential for automatic knowledge discovery. It discovers new insights through leveraging its existing knowledge only without further demonstration.

4.4 Performance v.s. documentation quality

We investigates the impact of documentation quality on performance. To assess LLM's capability to comprehend realistic documentation, we refrain from engineering or curating the content of the documentation. Instead, we vary the document length by truncating the documents and keeping the first n words, using it as a proxy for assessing thoroughness and quality. In this ablation, we consider the LLM-Cloud benchmark, which has long documentation based on real-world GCP CLI manuals. We illustrate the result in Figure 7.

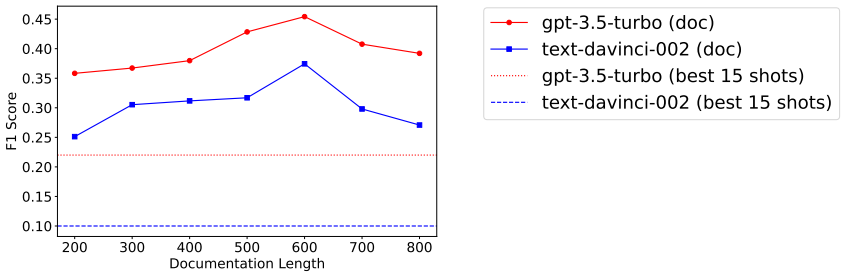

Figure 7: Performance of zero-shot documentation LLM when varying the input document length.

In both `text-davinci-002` and `gpt-3.5-turbo` experiments, we consistently observe a trend where performance improves as the document length increases, up to a length of 600. This finding aligns with our hypothesis that the models possess the ability to comprehend and leverage documentation effectively. Remarkably, this improvement in performance is achieved without any additional training, fine-tuning nor document curation . It highlights the tremendous value of providing comprehensive documentation, as it empowers the models to leverage a wide range of command-line tools *at scale*, solely through the process of reading and understanding the documentation.

We note that a degradation in performance after the document length exceeds 600 words. We attribute this decline to the inherent challenges associated with comprehending lengthy documents in language models [61]. However, we foresee the ongoing advancements in handling long inputs in language models will gradually address this limitation [10, 5, 2]. We leave exploring solutions for overcoming this limitation for future research.

5 Conclusion

In this paper, we examined the effectiveness of tool docs in enabling zero-shot tool usage with LLMs. We first showed that LLMs can achieve on par or better performance than their few-shot counterparts when provided with tool docs. We then scaled up to a significantly larger tool set on a newly collected API through docs only. By simply plugging in new tools along with their docs, LLMs are able to tackle unseen tasks in image editing and video tracking without further demos and replicate the functionalities of recent popular projects, suggesting a potential for automatic knowledge discovery. Overall, we shed light on a new perspective of tool usage with LLMs by focusing on their internal planning and reasoning capabilities with docs, rather than explicitly guiding their behaviors with demos.

References

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*, 2022.

[2] Anthropic. 100k context windows. https://www.anthropic.com/index/100k-context-windows, 2023. Accessed: 05/15/2023.

[3] AutoGPT. Auto gpt. https://autogpt.net/category/chatgpt-tools/autogpt/, 2023. Accessed: 05/15/2023.

[4] Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129, 1995.

[5] Amanda Bertsch, Uri Alon, Graham Neubig, and Matthew R Gormley. Unlimiformer: Long-range transformers with unlimited length input. *arXiv preprint arXiv:2305.01625*, 2023.

[6] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *International conference on machine learning*, pages 2206–2240. PMLR, 2022.

[7] SRK Branavan, David Silver, and Regina Barzilay. Learning to win by reading manuals in a monte-carlo framework. *Journal of Artificial Intelligence Research*, 43:661–704, 2012.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[9] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

[10] Aydar Bulatov, Yuri Kuratov, and Mikhail S Burtsev. Scaling transformer to 1m tokens and beyond with rmt. *arXiv preprint arXiv:2304.11062*, 2023.

[11] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*, 2023.

[12] Jiuhai Chen, Lichang Chen, Chen Zhu, and Tianyi Zhou. How many demonstrations do you need for in-context learning? 2023.

[13] 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.

[14] Ho Kei Cheng and Alexander G Schwing. Xmem: Long-term video object segmentation with an atkinson-shiffrin memory model. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXVIII*, pages 640–658. Springer, 2022.

[15] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[16] Surís Dídac, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. *arXiv preprint arXiv:2303.08128*, 2023.

[17] Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: An embodied multimodal language model. In *arXiv preprint arXiv:2303.03378*, 2023.

[18] 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.

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

[20] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR, 2020.

[21] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9147. PMLR, 2022.

[22] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

[23] IDEA-Research. Grounded-segment-anything. https://github.com/IDEA-Research/Grounded-Segment-Anything, 2023. Accessed: 05/15/2023.

[24] Shima Imani, Liang Du, and Harsh Shrivastava. Mathprompter: Mathematical reasoning using large language models. *arXiv preprint arXiv:2303.05398*, 2023.

[25] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. Mapping language to code in programmatic context. *arXiv preprint arXiv:1808.09588*, 2018.

[26] Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. UNIFIEDQA: Crossing format boundaries with a single QA system. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1896–1907, Online, November 2020. Association for Computational Linguistics.

[27] Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp. *arXiv preprint arXiv:2212.14024*, 2022.

[28] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*, 2022.

[29] Wonjae Kim, Bokyung Son, and Ildoo Kim. Vilt: Vision-and-language transformer without convolution or region supervision. In *International Conference on Machine Learning*, pages 5583–5594. PMLR, 2021.

[30] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[31] Mojtaba Komeili, Kurt Shuster, and Jason Weston. Internet-augmented dialogue generation. *arXiv preprint arXiv:2107.07566*, 2021.

[32] Angeliki Lazaridou, Elena Gribovskaya, Wojciech Stokowiec, and Nikolai Grigorev. Internet-augmented language models through few-shot prompting for open-domain question answering. *arXiv preprint arXiv:2203.05115*, 2022.

[33] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[34] Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. Visualbert: A simple and performant baseline for vision and language. *arXiv preprint arXiv:1908.03557*, 2019.

[35] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*, 2021.

[36] Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. Tapex: Table pre-training via learning a neural sql executor. *arXiv preprint arXiv:2107.07653*, 2021.

[37] Ruibo Liu, Jason Wei, Shixiang Shane Gu, Te-Yen Wu, Soroush Vosoughi, Claire Cui, Denny Zhou, and Andrew M Dai. Mind's eye: Grounded language model reasoning through simulation. *arXiv preprint arXiv:2210.05359*, 2022.

[38] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023.

[39] Pan Lu, Swaroop Mishra, Tony Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. In *The 36th Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[40] Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. *arXiv preprint arXiv:2304.09842*, 2023.

[41] Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. In *International Conference on Learning Representations (ICLR)*, 2023.

[42] Yujie Lu, Pan Lu, Zhiyu Chen, Wanrong Zhu, Xin Eric Wang, and William Yang Wang. Multimodal procedural planning via dual text-image prompting. 2023.

[43] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

[44] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[45] OpenAI. Gpt-4 technical report. 2023.

[46] Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*, 2023.

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

[48] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023.

[49] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

[50] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*, 2022.

[51] Yujia Qin, Shengding Hu, Yankai Lin, Weize Chen, Ning Ding, Ganqu Cui, Zheni Zeng, Yufei Huang, Chaojun Xiao, Chi Han, et al. Tool learning with foundation models. *arXiv preprint arXiv:2304.08354*, 2023.

[52] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023.

[53] Maxim Rabinovich, Mitchell Stern, and Dan Klein. Abstract syntax networks for code generation and semantic parsing. *arXiv preprint arXiv:1704.07535*, 2017.

[54] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[55] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[56] 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.

[57] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*, 2023.

[58] Kurt Shuster, Jing Xu, Mojtaba Komeili, Da Ju, Eric Michael Smith, Stephen Roller, Megan Ung, Moya Chen, Kushal Arora, Joshua Lane, et al. Blenderbot 3: a deployed conversational agent that continually learns to responsibly engage. *arXiv preprint arXiv:2208.03188*, 2022.

[59] Michael Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.

[60] Alane Suhr, Stephanie Zhou, Ally Zhang, Iris Zhang, Huajun Bai, and Yoav Artzi. A corpus for reasoning about natural language grounded in photographs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6418–6428, Florence, Italy, July 2019. Association for Computational Linguistics.

[61] Simeng Sun, Katherine Thai, and Mohit Iyyer. Chapterbreak: A challenge dataset for long-range language models. *arXiv preprint arXiv:2204.10878*, 2022.

[62] Romal Thoppilan, Daniel De Freitas, Jamie Hall, Noam Shazeer, Apoorv Kulshreshtha, Heng-Tze Cheng, Alicia Jin, Taylor Bos, Leslie Baker, Yu Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

[63] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[64] Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*, 2023.

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

[66] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*, 2022.

[67] 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 preprint arXiv:2201.11903*, 2022.

[68] Jerry Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, et al. Larger language models do in-context learning differently. *arXiv preprint arXiv:2303.03846*, 2023.

[69] Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual chatgpt: Talking, drawing and editing with visual foundation models. *arXiv preprint arXiv:2303.04671*, 2023.

[70] Jinyu Yang, Mingqi Gao, Zhe Li, Shang Gao, Fangjing Wang, and Feng Zheng. Track anything: Segment anything meets videos, 2023.

[71] Rui Yang, Lin Song, Yanwei Li, Sijie Zhao, Yixiao Ge, Xiu Li, and Ying Shan. Gpt4tools: Teaching large language model to use tools via self-instruction. *arXiv preprint arXiv:2305.18752*, 2023.

[72] Sherry Yang, Ofir Nachum, Yilun Du, Jason Wei, Pieter Abbeel, and Dale Schuurmans. Foundation models for decision making: Problems, methods, and opportunities. *arXiv preprint arXiv:2303.04129*, 2023.

[73] Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. Mm-react: Prompting chatgpt for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*, 2023.

[74] Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *arXiv preprint arXiv:2207.01206*, 2022.

[75] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[76] Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*, 2017.

[77] Wenhao Yu, Dan Iter, Shuohang Wang, Yichong Xu, Mingxuan Ju, Soumya Sanyal, Chenguang Zhu, Michael Zeng, and Meng Jiang. Generate rather than retrieve: Large language models are strong context generators. In *The Eleventh International Conference on Learning Representations*, 2023.

[78] Renrui Zhang, Jiaming Han, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, Peng Gao, and Yu Qiao. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*, 2023.

[79] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[80] Zhuosheng Zhang, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola. Multimodal chain-of-thought reasoning in language models. *arXiv preprint arXiv:2302.00923*, 2023.

[81] Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. In *International Conference on Machine Learning*, pages 12697–12706. PMLR, 2021.

[82] Victor Zhong, Tim Rocktäschel, and Edward Grefenstette. Rtfm: Generalising to novel environment dynamics via reading. *arXiv preprint arXiv:1910.08210*, 2019.

[83] Shuyan Zhou, Uri Alon, Frank F. Xu, Zhengbao Jiang, and Graham Neubig. Docprompting: Generating code by retrieving the docs. In *The Eleventh International Conference on Learning Representations*, 2023.

A Broader impacts and limitations

This work studies the importance of tool documentations in equipping LLMs with the ability to compose usages of a variety of tools to accomplish complex tasks. However, as discussed in [51], it is imperative to contemplate what tools should be made available to LLMs as well as how one should interpret and rely on the results obtained from the models. We envision tool documentations as a channel to guide LLMs in more safely using the tools, aligning with the original intended use of the tools.

B Implementation details

In this section, we provide further implementation details on each task. We conduct all our experiments on Debian GNU/Linux 10 machines with 40GB A100 GPUs.

B.1 ScienceQA

On ScienceQA [39], we closely follow the original setup [1] used in Chameleon [40], including the tool docs and few-shot demos (when used). We however find that the "Image Captioner" module used in the original work often provides less accurate captions on given images. In the documentation, we thus add the description on this observation for the "Image Captioner" module as shown in Figure 8.

```
The modules are defined as follows:

- Image_Captioner: This module generates a potentially inaccurate caption for the given
image. Avoid using the module unless necessary. "Image_Captioner" can be considered
when the question involves the semantic understanding of the image.

- Text_Detector: This module detects the text in the given image. Normally, we consider
using "Text_Detector" when the question involves the unfolding of the text in the image,
e.g., diagram, chart, table, map, etc., and the "has_image" field in the metadata is True.

- Knowledge_Retrieval: …
```

Figure 8: Documentations used in ScienceQA datasets. We used the original tool docs in Chameleon [40] and added the description for "Image Captioner" that the generated captions may be inaccurate.

B.2 TabMWP

On TabMWP [41], we strictly follow the original setup used in Chameleon [40]. We refer the readers to [40] and their open-sourced implementations for further details.

B.3 NLVRv2

On NLVRv2, we follow the setup used in [19]. However, as tool docs are not used in [19], we create our own docs for the tools used. Figure 9 shows the tool docs we use for several available tools used in VisProg [19].

[1] https://github.com/lupantech/chameleon-llm

```
Function: VQA
Description
The VQA function calls the BLIP Model for visual question answering. The model consists of a vision encoder, a text
encoder, and a text decoder. The vision encoder will encode the input image, the text encoder will encode the input
question together with the encoding of the image, and the text decoder will output the answer to the question.

Syntax
VQA(image: IMAGE, question: TEXT) -> TEXT or INTEGER or FLOAT or BOOL

Parameters
image: An IMAGE type input representing the image to be analyzed.
question: A TEXT type input representing the question to be answered about the image.

Returns
The function returns a TEXT, INTEGER, FLOAT or BOOL value, representing the answer to the input question about the
image.
The return variables would be INTEGER, FLOAT, or BOOL type when possible, otherwise it would be TEXT.

Use case:
Use VQA when you want to answer a question related to an image.

Example:
ANSWER1 = VQA(image=IMAGE1, question='What color is the car?')
ANSWER2 = VQA(image=IMAGE2, question='Does the image show a dog?')
ANSWER3 = VQA(image=IMAGE3, question='How many cups in the image?')


Function: EVAL
Description
The EVAL function calls the eval() function in Python. The expr argument is parsed and evaluated as a Python expression.
Variables can be expressed in {}. When evaluating expressions involving the results of other functions, such as VQA,
always use the EVAL function. The EVAL function also accepts the xor operator as the exclusive-or operator, which returns
true only when exactly one argument is true.

Syntax:
EVAL(expr: TEXT) -> TEXT or INTEGER or FLOAT or BOOL

Parameters:
expr: A TEXT type input representing a Python expression to be evaluated. The expression can include normal operators in
Python, as well as the additional xor operator for exclusive-or operat
ions.

Returns
The function returns a TEXT, INTEGER, FLOAT or BOOL value, representing the result of the evaluated Python
expression.

Use case:
Use EVAL when you want to evaluate a Python expression, especially when the expression involves results from other
functions.

Example
ANSWER0=EVAL(expr='{X} + 4 * 2 > 1 == False')
ANSWER1=EVAL(expr='{A} and {B} xor {C} or not {D}')
Important Note: When evaluating expressions involving the results of other functions, always use the EVAL function. For
example:

# Correct usage
ANSWER=EVAL(expr="{ANS}=='False' and {ANS2}=='False'")
FINAL_RESULT=RESULT(var=ANSWER)

# Incorrect usage
FINAL_RESULT=RESULT(var=ANS=='False' and ANS2=='False')
```

Figure 9: Example documentations used for tools in VisProg [19].

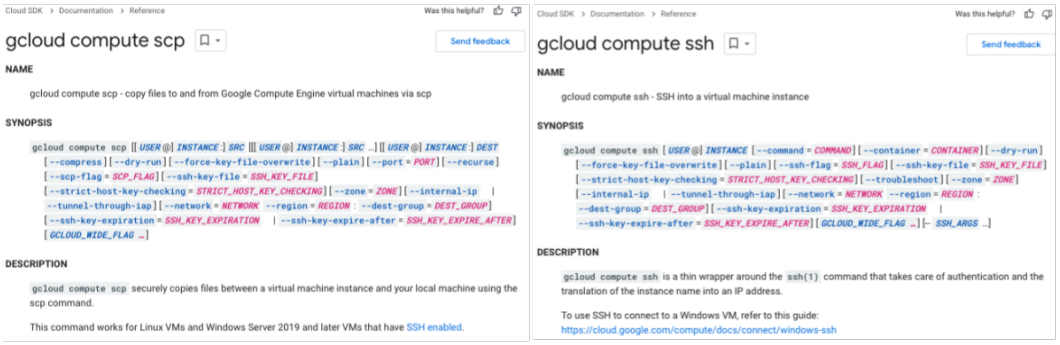

Figure 10: The documentation examples from GCP CLI. We crawl the website, remove the HTML tags and apply the renaming procedure as the documentation of the created LLM-Cloud CLI.

B.4 LLM-Cloud CLI

More examples. In Table 2, we show more examples of the created LLM-Cloud CLI dataset, based on GCP CLI.

Creating tool documentations. On the LLM-Cloud CLI dataset, we create tool documentations using the widely-used BeautifulSoup [2] library to scrape the GCP CLI documentation. We removed HTML tags and implemented the renaming procedures for LLM-Cloud CLI documentation. We note that we purposely do not eliminate unrelated content such as terms and hyperlinks. An example documentation from GCP before our renaming procedure is shown in Figure 10. This is to prevent excessive engineering of the documentation for better assessing the robustness of LLM documentation reading ability.

Documentation retrieval details. Given the extensive number of command-line tools in our experiments (200 in total), the complete documentation cannot fit within a single prompt. Consequently, for each query, we employ a simple TF-IDF search to retrieve the top 10 relevant documentations. We then truncate the length to a maximum of 600 words. We note that the actual token count depends on the tokenizer used by each LLM and is typically more than 600.

[2] https://pypi.org/project/beautifulsoup4/

Table 2: More examples of the created LLM-Cloud CLI dataset.

Question	**Commands in GCP**	**Commands after renaming (Final Answer)**
Show me how to deploy ocr-xer container and invoke it with a schedule every 2 hours on a project "test_proj" in sdk command lines. The ocr-xer container is located at "us-docker.pkg.dev/gcr-cleaner/ocr-xer/ocr-xer".	• gcloud config set project test_proj • gcloud run deploy ocr-xer --image=us-docker.pkg.dev/gcr-cleaner/ocr-xer/ocr-xer • gcloud scheduler jobs create http NAME --schedule --schedule="0 */2 * * *"	• llmcloud config set project test_proj • llmcloud run deploy ocr-xer --image=us-docker.pkg.dev/gcr-cleaner/ocr-xer/ocr-xer • llmcloud scheduler jobs make http NAME --schedule --schedule="0 */2 * * *"
How to deploy a machine learning model model.pt saved in my local to cloud via sdk command line?	• gsutil cp model.pt LOC/model.pt • gcloud ai-platform versions create VERSION --model MODEL --origin gs://LOC/model.pt	• llmutil cp model.pt LOC/model.pt • llmcloud ai-platform versions create VERSION --model MODEL --origin gs://LOC/model.pt
How to get transcript of a video test.mp4 at local via the cloud SDK?	• ffmpeg -i test.mp4 -ac 2 -f wav output.wav • gsutil cp test.wav LOC/test.wav • gcloud ml speech recognize-long-running --uri LOC/test.wav	• ffmpeg -i test.mp4 -ac 2 -f wav output.wav • llmutil cp test.wav LOC/test.wav • llmcloud ml speech recognize-long-running --uri LOC/test.wav
How to create a composer enviroment with a private ip network?	• gcloud composer environments create my_env • gcloud compute networks subnets update default --enable-private-ip-google-access	• llmcloud composer environments make my_env • llmcloud compute networks subnets update default --enable-private-ip-google-access
How to create a service account test@service.com with the name "AutoML" "BigQuery Data Editor" and ""AutoML Recommendations Service Account" permissions?	• gcloud iam service-accounts test@service.com --display-name AutoML • gcloud projects add-iam-policy-binding PROJ_ID --member="test@service.com" --role "roles/bigquery.dataEditor" • gcloud projects add-iam-policy-binding PROJ_ID --member "test@service.com" --role "roles/automlrecommendations.serviceAgent"	• llmcloud iam service-accounts test@service.com --display-name AutoML • llmcloud projects add-iam-policy-binding PROJ_ID --member="test@service.com" --role "roles/bigquery.dataEditor" • llmcloud projects add-iam-policy-binding PROJ_ID --member "test@service.com" --role "roles/automlrecommendations.serviceAgent"

B.5 Image editing and video tracking

As discussed in Section 4.3, by providing tool documentations, we can easily add on new tools to enable LLMs in solving novel tasks such as image editing and video tracking. Here, we leverage the recent advancements in vision models and expand the tool set used in VisProg [19] with three new tools: GroundingDINO [38], Segment Anything (SAM) [30], and XMem [14]. We provide their corresponding documentations in Figure 11.

```
Function: BETTERLOC
Description
The BETTERLOC function calls the GroundingDINO model to perform object localization. GroundingDINO is a zero-shot
text-conditioned object detection model. It returns all bounding boxes of the
 queried object. To make multiple queries at one time, separate different object names with ".".

Syntax
BETTERLOC(image: IMAGE, object: TEXT) -> BOX

Parameters
image: An IMAGE type input representing the image to be analyzed.
object: A TEXT type input representing the object to be localized in the image.

Returns
The function returns a BOX value, representing the bounding boxes of the queried object.

Use case:
Use BETTERLOC when you want to locate an object in an image and retrieve its bounding box(es).

Example:
BOX0 = BETTERLOC(image=IMAGE, object='cat')

Function: BETTERSEG
Description
The BETTERSEG function calls the Segment Anything Model (SAM) for image segmentation. It returns all objects detected
in the images as a list of OBJECT instances. Each OBJECT instance contain
s its bounding box and mask.

Syntax
BETTERSEG(image: IMAGE, box: BOX) -> LIST[OBJECT]

Parameters
image: An IMAGE type input representing the image to be analyzed.
box: The bounding boxes where we want to segment.

Returns
The function returns a LIST of OBJECT instances, each representing a detected object and including its bounding box, and
mask.

Use case:
Use BETTERSEG when you want to segment an object in a bounding box. Then the returned objects can be used by other
functions such as REPLACE, COLORPOP, BGBLUR.

Example:
BOX0 = BETTERLOC(image: IMAGE, object: 'fish')
OBJ0=BETTERSEG(image=IMAGE, box=BOX0)

Function: TRACK
Description
The TRACK function calls the XMem model for video object tracking. It takes an OBJECT instance from the first frame of
the video as input then returns all frames where the object is highlight
ed with a mask.

Syntax
TRACK(video: LIST[IMAGE], object: LIST[OBJECT]) -> LIST[IMAGE]

Parameters
video: A list of IMAGE type input representing the video to be analyzed.
object: The bounding boxes and masks of the objects which we want to track in the first frame of the video.

Returns
The function returns a list of a list of OBJECT instances representing the bounding boxes and masks of tracked objects in
each frame.

Use case:
Use TRACK when you want to track an object in a video. Then the returned list of objects can be used by other functions.

Example:
VIDEO0=TRACK(video=VIDEO, object=OBJ)

Important note: A video is a list of images. Use `IMAGE=EVAL(expr="{VIDEO}[i]")` in a separate line to get the i-th frame of
the video
```

Figure 11: Documentation of new tools introduced in VisProg. BETTERLOC, BETTERSEG, TRACK calls GroundingDINO, Segment Anything, XMem, respectively.

C Experimental results

In this section, we show the experimental results on each task with comparisons to more baselines.

ScienceQA. In Table 3, we compare zero-shot prompting with tool documentations to other baseline methods. We include the following baseline methods that are finetuned on the ScienceQA training set for performance reference: ViLT [29], VisualBERT [34], UnifiedQA CoT [39], MM-CoT [80], and LLaMA-Adapter [78]. We report the results obtained from [40] for the finetuned methods. For fair comparison, we shall focus on zero/few-shot settings. Thus, we include Chain-of-Thought (CoT) [67] and Chameleon [40] as the few-shot baselines to compare to. We see that with tool docs, we can not only achieve better performance than the few-shot methods without any demos, but we can also match (outperform) several models specifically finetuned on the dataset.

Table 3: Comparing zero-shot prompting with tool docs to existing baseline methods on ScienceQA. We see that zero-shot prompting with tool docs performs competitively, outperforming the two few-shot baselines and several finetuned models.

	Finetuned methods					Few-shot methods		Zero-shot methods
Benchmark	ViLT	VisualBERT	UnifiedQA CoT	MM-CoT	LLaMA-Adapter	CoT	Chameleon	0-shot with docs
ScienceQA	61.14	61.87	74.11	84.91	85.19	78.54	79.20	79.91

TabMWP. Similarly, in Table 4, we compare zero-shot prompting with tool docs to various finetuned models and few-shot baselines, inlcuding: UnifiedQA [26], TAPEX [36], Chain-of-Thought (CoT) [67], Program-of-Thought (PoT) [13], and Chameleon [40]. We report the results obtained from [40] for UnifiedQA, TAPEX, and CoT. We see that with tool docs, zero-shot prompting significantly outperforms finetuned models, and baseline few-shot methods, CoT and PoT. When compared to Chameleon that utilizes 16 few-shot tool-usage demos, tool docs enable the model to perform comparably without relying on any demos.

Table 4: Comparing zero-shot prompting with tool docs to existing baseline methods on TabMWP. We see that with tool docs, even zero-shot prompting without any tool-usage demos achieves better performance than finetuned models and few-shot CoT and PoT baseline. It also performs comparably to Chameleon that employs 16-shot tool-usage demos.

	Finetuned methods		Few-shot methods			Zero-shot methods
Benchmark	UnifiedQA	TAPEX	CoT	PoT	Chameleon	0-shot with docs
TabMWP	57.35	58.52	82.03	89.28	93.88	92.69

NLVRv2. In Table 5, we compare zero-shot prompting with tool docs to a finetuned model on NLVRv2 and various few-shot baselines. Specifically, we consider ViLT [29] as the finetuned baseline and VisProg [19] with varying numbers of tool-usage demos as the few-shot baselines. We report the result obtained from [19] for ViLT. Since VisProg does not utilize tool docs, we see that its performance is very sensitive to the number of demos used. In addition, we also observe large performance variances when we randomly select different demos used for prompting, e.g., the standard deviation for 2-shot prompting reaches 16.1 percentage point. This indicates that the few-shot demos may require careful curation for the model to achieve good performance. On the other hand, with tool docs, zero-shot prompting can already achieve decent performance compared to only using few-shot demos.

Table 5: Comparing zero-shot prompting with tool docs to existing baseline methods on NLVRv2.

	Finetuned methods	Few-shot methods				Zero-shot methods
Benchmark	ViLT	VisProg (0-shot)	VisProg (2-shot)	VisProg (4-shot)	VisProg (12-shot)	0-shot with docs
NLVRv2	76.30	0	43.1 ± 16.1	66.5 ± 1.4	69.1 ± 0.1	63.4

LLM Cloud-CLI. In Table 6, we present the results on LLM-Cloud CLI with different underlying LLM planners. On both `text-davinci-002` and `gpt-3.5-turbo`, when there is a large number of tools, we see documentation is much more important than few-shot demonstrations, where zero-shot with docs achieves significantly better performances than few-shot without docs. Additionally, when provided with docs, the LLMs are able to figure out how to use the tools without the need of demonstrations.

Table 6: Results on the LLM-Cloud CLI.

LLM	Number of Demos	Documentations	F1
`text-davinci-002`	0	No	0.02
	5	No	$0.02 \pm 0.02(0.05)$
	10	No	$0.05 \pm 0.02(0.11)$
	15	No	$0.05 \pm 0.05(0.1)$
	5	Yes	$0.35 \pm 0.02(\mathbf{0.37})$
	10	Yes	$0.35 \pm 0.02(\mathbf{0.37})$
	15	Yes	$0.35 \pm 0.01(\mathbf{0.37})$
	0	Yes	**0.37**
`gpt-3.5-turbo`	0	No	0.13
	5	No	$0.18 \pm 0.06(0.21)$
	10	No	$0.19 \pm 0.06(0.23)$
	15	No	$0.18 \pm 0.06(0.22)$
	5	Yes	$0.44 \pm 0.02(\mathbf{0.47})$
	10	Yes	$0.44 \pm 0.02(\mathbf{0.48})$
	15	Yes	$0.42 \pm 0.04(\mathbf{0.49})$
	0	Yes	**0.45**

Image editing. We provide more image editing examples achieved by zero-shot prompting with tool docs in Figure 12. In particular, we show that with tool docs, we are able to reproduce the image editing examples achieved by VisProg [19] without using any few-shot demos, wherein VisProg relies on 10 task-specific few-shot demos.

Original

Instruction: Replace man in black henley (person) with brick wall
Program:
OBJ0=SEG(image=IMAGE)
OBJ1=SELECT(image=IMAGE,object=OBJ0,query='man wearing a black henley',category='person')
IMAGE0=REPLACE(image=IMAGE,object=OBJ1,prompt='brick wall')
FINAL_ANSWER=RESULT(var=IMAGE0)

Instruction: Hide Salman and Aamir's faces with :ps, Shahrukh's faces with 8) and Hritik's with ;)
Program:
OBJ0=FACEDET(image=IMAGE)
OBJ1=SELECT(image=IMAGE,object=OBJ0,query='Salman Khan,Aamir Khan,Shahrukh Khan,Hritik Roshan',category='face')
OBJ2=SELECT(image=IMAGE,object=OBJ1,query='Salman Khan,Aamir Khan',category='face')
OBJ3=SELECT(image=IMAGE,object=OBJ1,query='Shahrukh Khan',category='face')
OBJ4=SELECT(image=IMAGE,object=OBJ1,query='Hritik Roshan',category='face')
IMAGE0=EMOJI(image=IMAGE,object=OBJ2,emoji='face_with_tongue')
IMAGE1=EMOJI(image=IMAGE0,object=OBJ3,emoji='smiling_face_with_sunglasses')
IMAGE2=EMOJI(image=IMAGE1,object=OBJ4,emoji='winking_face')
FINAL_ANSWER=RESULT(var=IMAGE2)

Instruction: Create a colorpop of the man in black henley and also blur the background
Program:
OBJ0=SEG(image=IMAGE)
OBJ1=SELECT(image=IMAGE, object=OBJ0, query='man wearing black henley', category='person')
IMAGE0=COLORPOP(image=IMAGE, object=OBJ1)
IMAGE1=BGBLUR(image=IMAGE0, object=OBJ1)
FINAL_ANSWER=RESULT(var=IMAGE1)

Figure 12: Image editing examples by zero-shot prompting `gpt-3.5-turbo` with tool docs. Zero-shot prompting with docs is able to reproduce the results achieved by VisProg using few-shot demos [19].