

GPT4Tools: Teaching Large Language Model to Use Tools via Self-instruction

Rui Yang^{1*†}, Lin Song^{2*†}, Yanwei Li³, Sijie Zhao², Yixiao Ge², Xiu Li¹, Ying Shan²

¹Tsinghua Shenzhen International Graduate School, Tsinghua University

²Tencent AI Lab ³Chinese University of Hong Kong

rayyang0116@gmail.com ronny.song@tencent.com

The essential difference between humans and animals is that humans are capable of making and using tools.

—Friedrich Engels

Abstract

This paper aims to efficiently enable Large Language Models (LLMs) to use multimodal tools. Advanced proprietary LLMs, such as ChatGPT and GPT-4, have shown great potential for tool usage through sophisticated prompt engineering. Nevertheless, these models typically rely on prohibitive computational costs and publicly inaccessible data. To address these challenges, we propose the GPT4Tools based on self-instruct to enable open-source LLMs, such as LLaMA and OPT, to use tools. It generates an instruction-following dataset by prompting an advanced teacher with various multi-modal contexts. By using the Low-Rank Adaptation (LoRA) optimization, our approach facilitates the open-source LLMs to solve a range of visual problems, including visual comprehension and image generation. Moreover, we provide a benchmark to evaluate the ability of LLMs to use tools, which is performed in both zero-shot and fine-tuning ways. Extensive experiments demonstrate the effectiveness of our method on various language models, which not only significantly improves the accuracy of invoking seen tools, but also enables the zero-shot capacity for unseen tools. The code and demo are available at <https://github.com/StevenGrove/GPT4Tools>.

1 Introduction

Recent advances in large language models (LLMs), such as GPT-3 [1], InstructGPT [2], and ChatGPT [3], have demonstrated substantial potential in the area of zero-shot learning and logical reasoning. These models are typically trained on a large volume of text-only data, primarily sourced from the internet. However, as promising as they may seem, these advanced proprietary LLM [3, 4] have significant limitations. One of the major hindrances is the high computational cost associated with these models, which may not be affordable or accessible to many scenarios. Additionally, these models typically depend on specialized data, such as source code and conversation history, which are not easily available to the public.

Instead of solely focusing on language processing, many recent researches [5, 6] attempt to bridge the gap between language models and multi-modal tools. The intelligent agents like Visual ChatGPT [5] and MMREACT [6] have made efforts to meet this goal by sophisticated prompt engineering. These agents utilize a pre-defined template to create instructions that can be executed by vision-language foundation models. Although these approaches have led to impressive results, the primary process of instruction decomposition is heavily based on GPT-3.5 [3], which is not publicly available, thus

*Equal contribution. † Work done during an internship at Tencent.

†Corresponding author.

Table 1: Comparison of related works. ‘LM’ is the language models. ‘Mechanism’ denotes how the language model learns to invoke tools. ‘Unseen’ indicates the zero-shot capability on unseen tools.

Method	LM	Mechanism	Teacher	Multi-Modal	Unseen
Lazaridou et al. [10]	Gopher-280B [14]	prompt	✗	✗	✗
ToolFormer [11]	GPT-J (6B) [15]	self-instruct	✗	✗	✗
Visual ChatGPT [5]	GPT-3.5 (175B) [3]	prompt	✗	✓	✓
MMREACT [6]	GPT-3.5 (175B) [3]	prompt	✗	✓	✓
GPT4Tools (ours)	Vicuna-13B [12]	self-instruct	✓	✓	✓

limiting further advancements. In addition, equipping these agents with the capability to use tools requires a large amount of data [3]. This brings up an open question: *how to efficiently enable a primitive language models to use multi-modal tools?*

To achieve it, different from previous studies [7–11], we explore a new perceptive as illustrated in Table 1. We propose a simple yet effective method, called GPT4Tools, designed to empower open-source LLMs with the ability to use tools via self-instruct from advanced LLMs. To achieve this, we construct an instruction dataset by prompting advanced teachers (for example, ChatGPT [3]) conditional on visual content and tool descriptions, which leads to the generation of tool-related instructions. Unlike Toolformer [11], our method can utilize visual content description to significantly improve data diversity. Furthermore, with the generated instruction-following dataset, we incorporate Low-Rank Adaptation (LoRA) to fine-tune the primitive language models, such as Vicuna [12] and OPT [13]. Besides intrinsic language abilities, by using GPT4Tools, language models can also have the capability to use tools to solve a variety of visual problems. The tasks include visual comprehension and image generation, such as object grounding and segmentation, generating and instructing images, and visual question answering (VQA). With the proposed GPT4Tools, the LLMs not only significantly improves the accuracy of invoking seen tools, but also enables the zero-shot capacity for unseen tools in a zero-shot manner.

We propose an evaluation metric to assess the effectiveness of LLMs in utilizing tools across diverse tasks. With this metric, two human-curated validation sets are constructed to evaluate the LLMs in zero-shot and fine-tuning ways, providing a comprehensive measure of the ability to use tools. To demonstrate the effectiveness of GPT4Tools, we conduct extensive experiments on various language models. The results show the efficacy in teaching LLMs when and how to use tools. Specifically, with the GPT4Tools, the fine-tuned Vicuna-13B achieves 9.3% absolute gains in successful rate over GPT-3.5 [3], which acquires tool priors in context. Moreover, the fine-tuned Vicuna-13B shows the strong capacity to invoke unseen tools, which is comparable to GPT-3.5 in successful rate.

Our GPT4Tools stands distinct from previous and concurrent studies [5–11] in three ways. First, our method enables primitive open-source language models to use tools, eliminating the dependence on advanced proprietary LLMs like ChatGPT. Second, we design a new approach based on multi-modal contexts for self-instruction and augmentation, which significantly promote the multi-modal tool usage and can be deployed in different approaches. Third, we propose a new benchmark to assess the effectiveness of using tools, and our method shows remarkable improvements.

2 Related Work

Vision and Language Model. In the quest to achieve multimodal models capable of addressing both language and vision tasks, several studies [16–21] have explored methods to enable language models to comprehend visual input. These include techniques such as transforming images into discrete textual representations [16, 17] or projecting continuous image features into the textual feature space [22–25]. Concurrently, other research has been dedicated to the development of generalist models [18, 19, 21, 20], which permit a model to simultaneously input images and text, eliminating the necessity for a projection process. For instance, OFA [26] devised a unified sequence-to-sequence decoding architecture applicable to language and object detection tasks. Similarly, Pixel2Pixel [20] converted the outcome of visual comprehension tasks into a series of discrete tokens akin to language tasks. Gato [21] brought together a range of vision and control tasks into a sequential prediction issue, while UViM [27] and Unified-IO [19] advocated for the learned discrete codes as a means to unify an

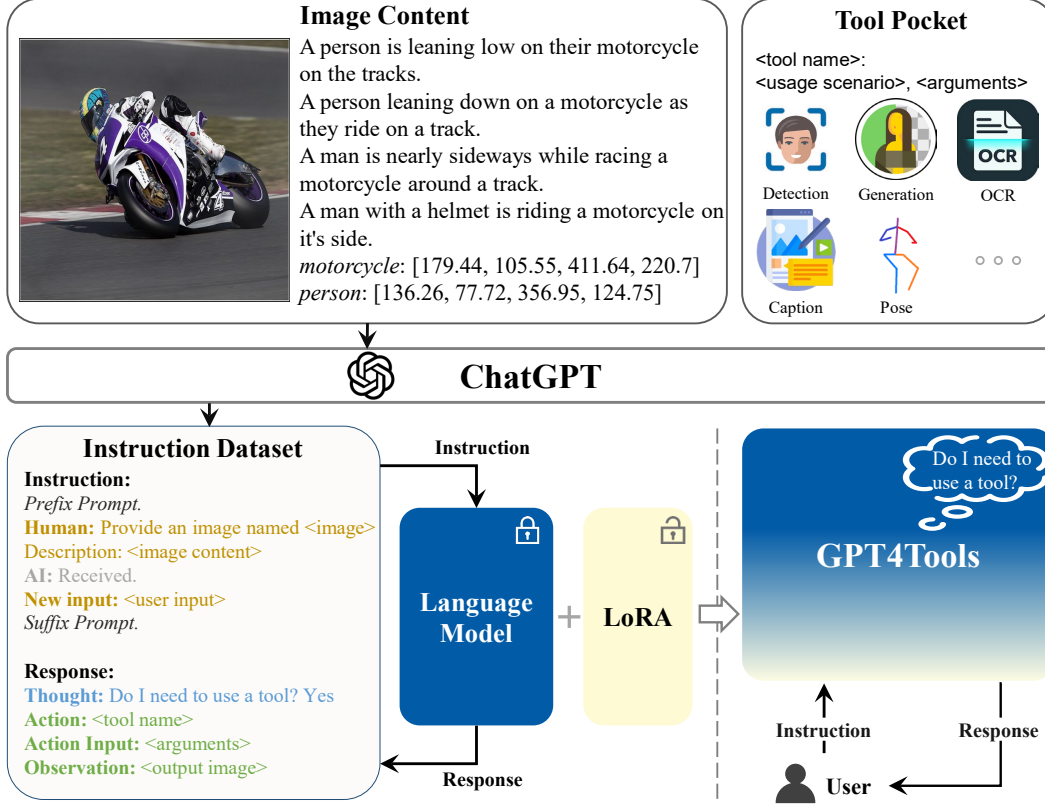


Figure 1: Diagram of the GPT4Tools. We prompt the ChatGPT with image content and definition of tools in order to obtain a tool-related instruction dataset. Subsequently, we employ LoRA [38] to train an open source LLM on the collected instruction dataset, thus adapting the LLM to use tools.

array of vision tasks. By contrast, we in this paper equip the language model with a diverse array of specialized multi-modal tools to process distinct vision tasks. This approach not only promotes the scalability of the model for various tasks, but also avoids the issue of forgetfulness stemming from repeated fine-tuning.

Instruction Tuning. Recent studies [28, 2, 29–32] have turned out that pre-trained language models could follow natural language instructions and complete various real-world tasks if they are tuned on specific instruction-following data. Notably, InstructGPT [2], FLAN-T5 [30], OPT-IML [32] demonstrated remarkable performance on specific tasks after being fine-tuned with instruction data. In order to release the cost of human-written instructions, Self-Instruction [33] found that the instruction-following capabilities of language models can be enhanced by turning on their own generated instruction data. More importantly, this approach inspired a feasible means to improve the zero- and few-shot abilities of language models, i.e., distilling off-the-shelf language models using instructional data from strong ChatGPT [3] or GPT-4 [4]. As a result, many recent works [12, 34–36] tried to construct excellent language models for various applications based on the LLaMA [37]. For instance, Stanford-Alpaca has employed 52K instructions generated by GPT-3.5 [3] to construct an exceptional dialogue model. LLaVa [36] has adopted GPT-3.5 [3] and GPT-4 [4] to incorporate instruction-following data related to visual content. In this paper, we use GPT-3.5 [3] to construct tool-related instruction datasets, thereby allowing other language models to acquire tool usage capabilities.

Tool Usage. In the Natural Language Processing (NLP) community, several arts [7–11] sought to endow language models with the ability to use tools. For instance, Komeili et al. [7] proposed to generate conversation responses conditioned on the results of the search engine. LaMDA [9] created a set of tools (comprising an information retrieval system, a calculator, and a translator) to avoid plausible outputs. Lazaridou et al. [10] utilized few-shot prompting on Gopher-280B [14] to enable the search engine to ground its output in factual and current information. Similarly, Visual ChatGPT [5] and MMREACT [6] prompted ChatGPT to invoke visual foundation models. In addition,

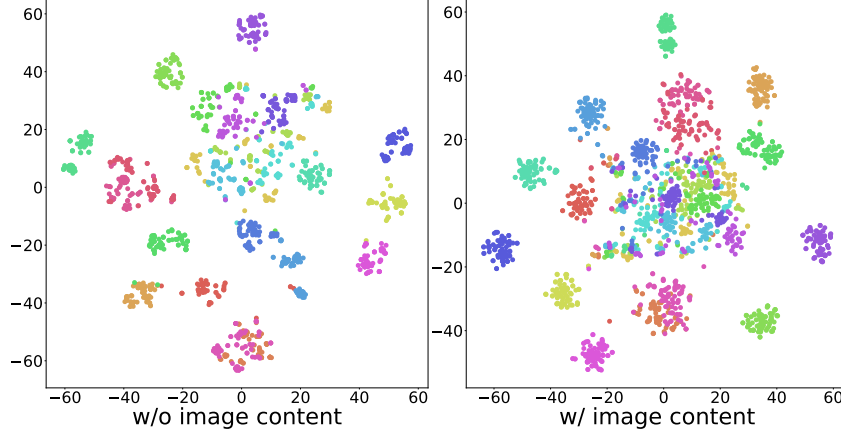


Figure 2: t-SNE¹ visualization for instruction data with and without image content.

ToolFormer [11] used self-instruction and bootstrapping to teach GPT-J (6B) [15] using five tools, which include a question and answer system, a calculator, a search engine, a machine translation system, and a calendar. On the contrary, we focus on using the GPT-3.5 model as a powerful teacher to distill off-the-shelf language models and enable them to access many visual models.

3 Method

Large language models (LLMs) [1, 13, 14] have shown remarkable in-context learning abilities. Among them, ChatGPT [3] and GPT-4 [4] are proven to effectively perform text-annotation tasks [39] or instruct other models to follow instructions of specific domains [40, 34, 12, 36]. Inspired by these findings, we propose leveraging ChatGPT as a powerful teacher to enable off-the-shelf language models to acquire tool usage capabilities. Specifically, we utilize ChatGPT to generate tools-related instruction-following data, which is then used to tune the language model. This process enables the language model to access multimodal information by invoking visual models. Furthermore, we propose an evaluation metric to assess the tool-use ability of the given language model. In the following, we present the data generation, instruction tuning, and evaluation metric in turn.

3.1 Dataset Construction

Data Generation. Figure 1 illustrates the process of generating tool-related instruction dataset. Given an image, we construct the image content X_C according to the captions and bounding boxes, which is a straightforward means of establishing connections between an image and a language model [16, 17]. Conditioned upon the X_C , we provide the ChatGPT [3] (M_T) with a tool-related prompt P_t whereby attaining a large number of instruction-following data:

$$Y \sim M_T(P_t|X_C). \quad (1)$$

The P_t comprises the system message, the definition of tools (`<tool name> : <usage scenario>, <arguments>`), and suffix prompt which encourage M_T to generate visual instructions and desired outputs. Y , the outcome of M_T , consists of N instruction-output pairs $\{y^1, y^2, \dots, y^N\}$, where y_i has the format of "`<instruction>, <tool name>, <arguments>`", and N is the number of defined tools. As each input of M_T is grounded to the image content X_C , the generated instructions are inherently connected to the image, thus avoiding arbitrary generation. Moreover, the rich variability of the image results in a higher diversity of instructions when compared to imagined ones. To provide contrast, we also collect instruction follow-up data without image content X_C , which is similar to ToolFormer [11]. As depicted in Figure 2, the high degree of similarity observed among instructions generated without X_C highlights their monotony and lack of diversity. On the contrary, instructions generated with image-conditioned prompts are more informative and diverse, due to changes in the image content.

¹The visual instruction and tool arguments are embedded by Sentence-BERT [41]. During generation without image content, we set the temperature as 0.9 to avoid constant outcomes.

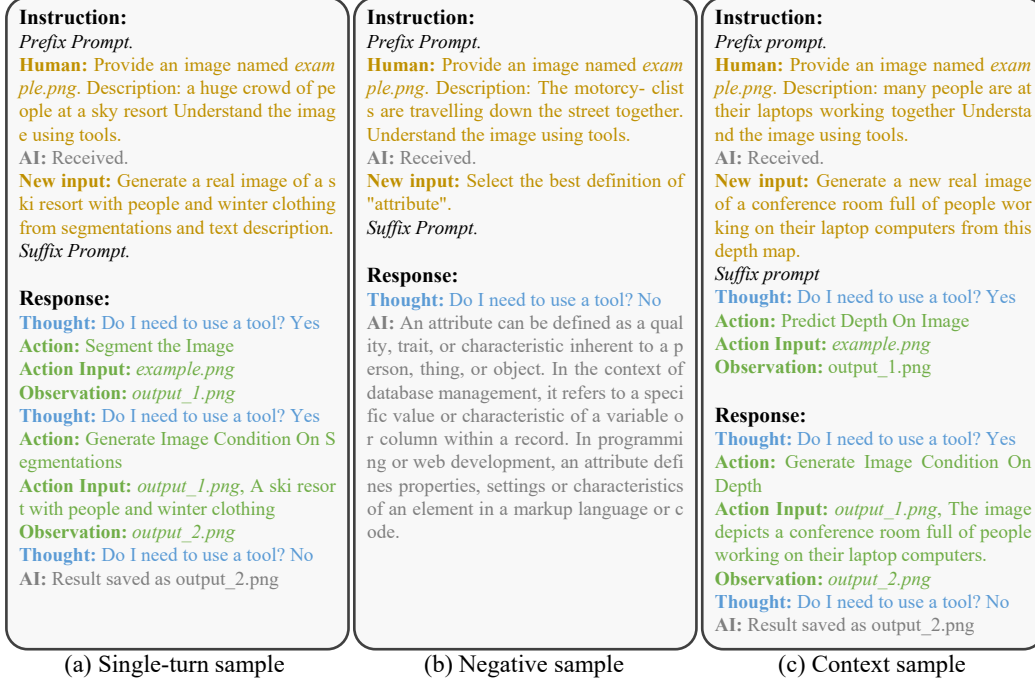


Figure 3: Samples of the single-turn instruction, negative instruction, and contextual instruction.

Data Formation. Upon the collected raw dataset (70K items), we apply a filtering process to remove similar instructions, resulting in 41K retained items. Subsequently, we transform the retained data into an instruction-response format utilizing a standardized template as shown in the bottom-left corner of Figure 1. This procedure produces a new dataset, denoted as Y_S^+ . The instruction component of Y_S^+ incorporates a *prefix prompt* that encompasses system messages and tool definitions, `<image content>` that denotes the image content, `<user input>` that is replaced with the generated visual instruction, and a *suffix prompt* designed to prompt the language model to reply the user input through given tools. The response component of Y_S^+ comprises the *Thought* to invoke tools and a chain of actions. Each action involves an *Action* and *Action Input*, succeeded by `<tool name>` and `<arguments>`, respectively. The *Observation* reflects the outcome of the invoked tools. A sample from Y_S^+ is presented in the Figure 3 (a).

Data Augmentation. Although we have successfully acquired instruction-following data related to the tool usage, this simplistic format lacks complexity and depth in both instructions and responses. To mitigate this issue, we augment the generated data from two perspectives:

- *Negative samples.* The generated instructions primarily focus on tool usage, i.e., the decision after the *Thought* is always "Yes". Consequently, there is a potential risk that the fine-tuned model overfits such a decision. When the user instruction does not connect with the tool usage, the fine-tuned model may erroneously execute irrelevant actions by invoking unnecessary tools. To mitigate this issue, we synthesize negative samples Y_S^- by selecting conversation data from the existing dataset [40] and converting them into the required template, as illustrated in Figure 3 (b). By tuning the model with $Y_S^+ \cup Y_S^-$, it can decide when to use tools.
- *Context samples.* The generated instructions adopt a standard and fixed single-tune format, which lacks a contextual structure. Thus, we augment the dataset by cutting off the chain of action, as shown in Figure 3 (c). Furthermore, we randomly select multiple instructions from $Y_S^+ \cup Y_S^-$ and reformat them into multi-turn conversation data. In this way, we synthesize the contextual instruction-following data Y_S^c , which enables the tuned model to call tools within the given context.

So far, we have constructed the tool-related instructional dataset, including positive samples, negative samples, and context samples: $Y_S = Y_S^+ \cup Y_S^- \cup Y_S^c$.

3.2 Instruction Tuning

Based on the Y_S , we tune the off-the-self language model using its original auto-regressive training objective. To make the tuning feasible, we leverage LoRA [38] optimization, which freezes the language model and optimizes rank decomposition components of the Transformer layers. For a sequence with L tokens, we compute the probability of the target response X_r by:

$$p(X_r|X_C, X_{inst}) = \prod_{i=1}^L p_\theta(x_i|X_C, X_{inst}, x_{1:i-1}), \quad (2)$$

where X_{inst} denotes the instruction tokens; and θ is the trainable parameters. In practice, *prefix prompt* and *suffix prompt* are also involved, but we here skip them for better readability.

3.3 Evaluation Approach

Numerous benchmarks [42, 43, 8] typically utilize human-annotated datasets to evaluate the performance of a model. For the purpose of measuring the tool-usage capacity of the language model, we construct a evaluation dataset following the same procedures detailed in § 3.1, and manually verify the accuracy of each constituent item. This evaluation dataset is partitioned into two components: the first part (validation set) has the same ingredients as the training set, encompassing 23 tools; the second part (test set) comprises 8 novel tools that are absent from the training set. We will use the validation set to validate whether the model can adhere to user commands correctly after tuning with the training set. The test set will be employed to verify whether the model can generalize to new tools after tuning. Based on the human-annotated evaluation dataset with N instructions, we design a successful rate to measure the model’s performance from three aspects:

- **Successful Rate of Thought** (SR_t) measures whether the predicted decision matches the ground-truth decision. It is calculated as $SR_t = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(\tau_i)$, where τ_i signifies a singular process. If the thought is correct, $\mathbb{I}(\tau_i)$ is equal to 1, and 0 otherwise.
- **Successful Rate of Action** (SR_{act}) measures whether the predicted tool name is in agreement with the name of the ground truth tool. It is calculated as $SR_{act} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(\alpha_i)$, where α_i denotes the matching process for the tool names. In cases where the predicted tool name matches the pre-defined name, $\mathbb{I}(\alpha_i)$ is equal to 1, and 0 otherwise.
- **Successful Rate of Arguments** (SR_{args}) evaluates whether the predicted arguments match the ground-truth arguments. It can be calculated using the following equation:

$$SR_{args} = \frac{1}{N} \sum_{i=1}^N \eta_i, \text{ where } \eta_i = \frac{1}{K} \sum_j^K \eta_{i,j}. \quad (3)$$

Here, η_i denotes a sequence of arguments, encompassing both the image path and the input text. For instance, ControlNet [44] needs the image path saved conditions (e.g. pose map, depth map, or segment map) and the input text described the user command. K represents the quantity of arguments in η_i . When the argument belongs to the image path, $\eta_{i,j}$ equals 1 if the predicted and ground-truth image paths share the same suffix, and 0 otherwise. When the argument is the input text, $\eta_{i,j}$ is equal to the BLEU score between the predicted and the ground truth text.

- **Successful Rate** (SR) measures whether a chain of actions are executed successfully, which requires the correctness of thought, tool name, and tool arguments:

$$SR = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(\tau_i) \cdot \mathbb{I}(\alpha_i) \cdot \mathbb{I}(\eta_i > 0.5) \quad (4)$$

Additionally, when a procedure comprises two consecutive actions, the SR equals 100% only if both actions are executed correctly.

4 Experiments

4.1 Implementation Details

We employ the ChatGPT (gpt-3.5-turbo) [3] as the teacher model to generate the raw instruction-following data. Since this study focused on teaching off-the-self language models to use tools instead

Table 2: Comparison of different language models. The zero-shot prediction is adopted for unseen tools and the models without GPT4Tools.

Model	GPT4Tools	Validation (seen tools)				Test (unseen tools)			
		SR _t	SR _{act}	SR _{args}	SR	SR _t	SR _{act}	SR _{args}	SR
GPT-3.5 [3] (text-davinci-003)	✗	93.5	96.1	78.0	84.8	99.5	99.5	91.5	91.5
OPT-13B [13]	✗	1.1	1.2	0.0	0.0	0.0	0.0	0.0	0.0
	✓	99.4	98.3	89.2	93.2	97.8	89.6	84.0	78.6
LLaMa-13B [37]	✗	20.4	15.7	16.5	3.2	16.1	17.6	21.7	2.0
	✓	77.3	74.9	71.4	66.4	74.2	72.2	70.9	69.9
Vicuna-13B [12]	✗	69.2	25.1	25.2	12.4	84.4	43.7	46.7	26.2
	✓	98.7	97.6	91.4	94.1	98.2	97.0	92.2	90.6

of prompt engineering, we adopted the methodology outlined in the Visual ChatGPT [5] to construct tool-related prompts. Our tool pocket consists of 31 tools, including the 23 tools defined in Visual ChatGPT [5] and 8 additional tools (refer to Table 6 in Appendix B for detailed tool names). The training set comprises 71K instruction-response pairs, wherein all instructional data is related to the 23 tools from Visual ChatGPT. We divided the human-annotated evaluation dataset into two parts: validation set and test set. The validation set contains the same tools as the training set, with approximately 50 items associated with each tool. The test set includes tools that are not present in the training set. (further details provided in Appendix A)

Based on the collected data, we tuned language models (LLaMA [37], Vicuna [12], and OPT [13]) with LoRA [38] technology. Specifically, we equipped the projection layers of query, key, value, and output with LoRA layers. The LoRA attention dimension and scaling alpha were set to 16. While the language model was kept frozen, the LoRA layers were optimized using the AdamW [45]. All models were fine-tuned over 3 epochs, with a batch size of 512. The learning rate was set to 3×10^{-4} , and the maximum length of new tokens was restricted to 2048. Unless otherwise specified, we used Vicuna-13B for the ablation experiments.

4.2 Main Result

Can instruction datasets teach language model using tools? The outcomes of GPT-3.5 [3], OPT-13B [13], LLaMA-13B [37], and Vicuna-13B [12] are presented in Table 2. GPT-3.5 is considered analogous to Visual ChatGPT [5]. Upon prompting GPT-3.5 with tool-associated instructions, it is able to attain a SR of 84.8% on the validation set, thereby underscoring its zero-shot ability to follow a standardized format and utilize tools effectively. Notably, OPT-13B fails to invoke tools with the prompts alone. In contrast, LLaMA-13B and Vicuna-13B exhibit a certain level of comprehension of tool usage, while they still face challenges in executing a chain of actions. Specifically, LLaMA-13B achieves 3.2% SR, which is absolutely lower than SR_t, SR_{act}, and SR_{args}. In the case of Vicuna-13B, its SR is 56.8% less than SR_t, implying that under a zero-shot setup, Vicuna-13B displays commendable discernment in determining when to use tools within a given context. After fine-tuned with GPT4Tools, there are substantial alterations in the tool invocation competencies of each model. Specifically, the SR of OPT-13B is witnessed a sharp increase from 0 to 93.2%. Similarly, the SR for LLaMA-13B escalates from 3.2% to 66.4%, and Vicuna-13B’s SR rises from 12.4% to 94.1%. These outcomes unequivocally validate that the GPT4Tools developed in this study are indeed effective in instructing language models to use tools.

Can the model be generalized to unseen tools after fine-tuning? The right side of Table 2 shows the results when prompting a novel tool and corresponding utilization. On the test set, GPT-3.5 attains 91.5% SR in a zero-shot manner. The outcomes for other models, which are not fine-tuned on the GPT4Tools and directly invoke tools utilizing prompts, are analogous to those on the validation set. In contrast, models that are fine-tuned on the GPT4Tools dataset exhibit a degree of competence in invoking tools that have not been previously encountered (did not appear in the training set). More specifically, the fine-tuned LLaMA-13B model achieves a superior SR on new tools by a margin of 67.9% when compared to the original model. The fine-tuned Vicuna-13B model demonstrates 90.6%

Table 3: Ablation study for data augmentations on the validation set. 'IC', 'CS', 'NS' denotes image content, context samples, and negative samples, respectively.

IC	CS	NS	SR_t	SR_{act}	SR_{args}	SR
			70.0	55.7	51.7	36.9
✓			89.6	89.9	84.5	81.6
✓	✓		97.4	95.7	88.5	91.6
✓	✓	✓	98.7	97.6	91.4	94.1

Table 4: Ablation study for different model sizes on the validation set.

Model	GPT4Tools	SR_t	SR_{act}	SR_{args}	SR
Vicuna-7B [12]	✗	27.7	15.8	11.5	4.5
	✓	96.2	94.5	89.8	92.9
Vicuna-13B [12]	✗	69.2	25.1	25.2	12.4
	✓	98.7	97.6	91.4	94.1

SR on new tools, which is comparable to GPT-3.5. This observation indicates that the language model can invoke unseen tools after fine-tuned with GPT4Tools.

4.3 Ablation Study

Data Augmentation. As depicted in Table 3, we execute a series of ablation studies on various tricks implemented during the creation of the dataset. When instructions are not conditioned upon the image content, the SR of the fine-tuned model on the validation set is a mere 36.9%. In contrast, when instructions are generated with conditioning on the image content, the SR on the validation set is enhanced substantially to 81.6%. This uptick can be primarily attributed to the elevated diversity and intricacy of the generated instructions. Moreover, an augmentation of the SR to 91.6% is observed upon the introduction of context samples into the instructions. This finding underscores the fact that partitioning the chain of actions and allocating them to the instruction and response can strengthen the model’s comprehension of the tool. It is noteworthy to mention that with the incorporation of negative samples into the generated instructions, the SR increases to 94.1%. This outcome can be traced back to the propensity of the model, when trained exclusively with positive samples, to bias toward tool invocation. This tendency consequently diminishes the capacity to discern the appropriate cases for tool usage. Adding negative samples equips the model with the ability to determine when to use tools.

Model Size. We attempt experiments with models at different scales. The results in Table 4 demonstrate that after fine-tuned on the generated dataset, Vicuna-7B [12] is also capable of invoking tools in a fixed format. Specifically, under a zero-shot setting, Vicuna-7B achieves only a 4.5% SR. By contrast, after fine-tuning, it is able to achieve an SR of 92.9%.

Tuning Iteration. We increase the number of iterations for fine-tuning and present the results in Figure 4. Notably, during the range of iterations from 400 to 800, the model’s performance demonstrates substantial fluctuations in tool invocation. However, subsequent to this range, there is a steady improvement in SR_t , SR_{act} , SR_{args} , and SR. This indicates that the model progressively adapts to the dataset and enhances its capability to invoke tools.

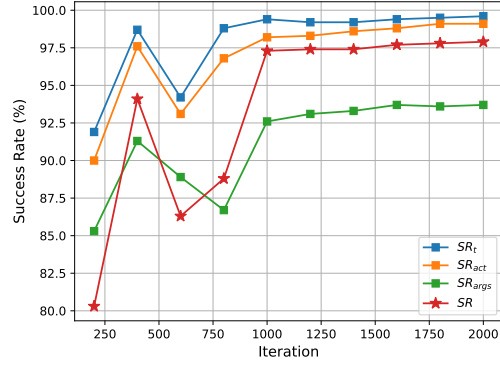


Figure 4: Performance variation curve with the fine-tuning iteration.

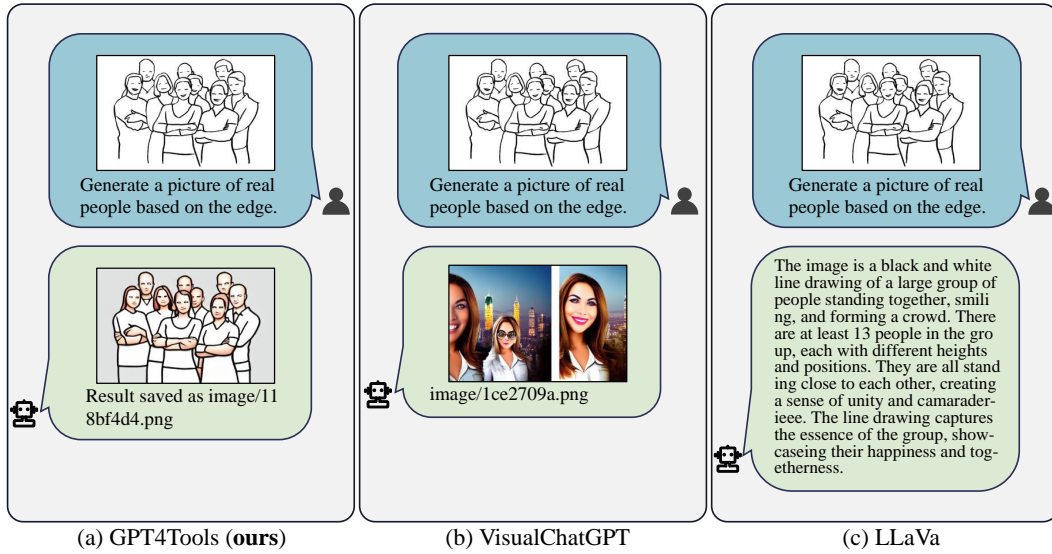


Figure 5: Comparison with other models. Our GPT4Tools responses correctly, while Visual ChatGPT [5] replies with the wrong image, and LLaVa [36] can not generate the image.



Figure 6: Cases of invoking tools from Vicuna-13B [12] fine-tuned on our GPT4Tools.

4.4 Case Study

Figure 5 presents a comparative analysis of our model with Visual ChatGPT [5] and LLaVa [36]. When an image is submitted by the user alongside the instruction "Generate a picture of real people based on the edge", Visual ChatGPT delivers an image that exhibits a weak correlation with the given instruction. Owing to its inability to generate images, LLaVa only returns a caption. In contrast, our model produces an accurate result, thereby evidencing that the tool-related instruction tuning method proposed in this paper can effectively instruct language models in the correct usage of tools. In the Figure 6, we further demonstrate that the Vicuna-13B fine-tuned on GPT4Tools is capable of finishing some visual commands by invoking visual tools. This finding indicates that imparting knowledge to language models regarding the tool invocation could potentially be a way toward the development of a generalist model. More case studies are present in Appendix C.

5 Limitation

Although the proposed GPT4Tools can teach plug-and-play language models to use tools effectively, it still has some limitations. For instance, the success rate of all models is not 100%, thus further improvements are still necessary for practical applications. Additionally, GPT4Tools teaches the model to explicitly invoke tools using a verbose and fixed prompt (Table 9). This approach reduces the computational efficiency of the model as attention-based architectures compute the relationships between all tokens. Therefore, in the future, it should be explored how to enable the model to implicitly invoke various tools, rather than using the complex prompt. Nevertheless, our GPT4Tools provides a viable approach for equipping language models with multimodal tools.

6 Conclusion

This paper introduces GPT4Tools, a novel method that enables open-source LLMs to utilize multimodal tools efficiently. We construct a tool-related instructional dataset from advanced ChatGPT and augment them by introducing negative and context samples. Based on the built dataset, we employ LoRA optimization to enhance LLMs' tool-usage capabilities, thus allowing LLMs to handle various visual tasks, e.g., visual comprehension and image generation. Moreover, we propose a benchmark to assess tool usage accuracy from the decision when to use tools, which tools to use, and arguments of invoked tools. In this benchmark, the LLMs tuned with our GPT4Tools perform comparably to GPT-3.5 on unseen tools. We desire the GPT4Tools to pave the way for one thing, i.e., to equip LLMs with multimodal tools.

A GPT4Tools Dataset

Table 5: Summary of tool names. Gray tool names are from Visual ChatGPT [5]. Black tool names are new in GPT4Tools.

Image Generation	Image Understanding	
Generate Image From User Input Text	Detect the Given Object	Text Detection On Image
Generate Image Condition On Canny Image	Segment the Image	Detection
Generate Image Condition On Depth	Get Photo Description	Image Super-Resolution
Instruct Image Using Text	Edge Detection On Image	Crop the Given Object
Generate Image Condition On Sketch Image	Predict Depth On Image	Assess the Image Quality
Replace Something From The Photo	Line Detection On Image	Recognize Face
Generate Image Condition On Segmentations	Answer Question About The Image	Detect Face
Generate Image Condition On Pose Image	Sketch Detection On Image	
Generate Image Condition On Soft Hed Boundary Image	Pose Detection On Image	
Generate Image Condition On Normal Map	Hed Detection On Image	
Remove Something From The Photo	Predict Normal Map On Image	
Generate 3D Asset From User Input Text	Segment the Given Object	

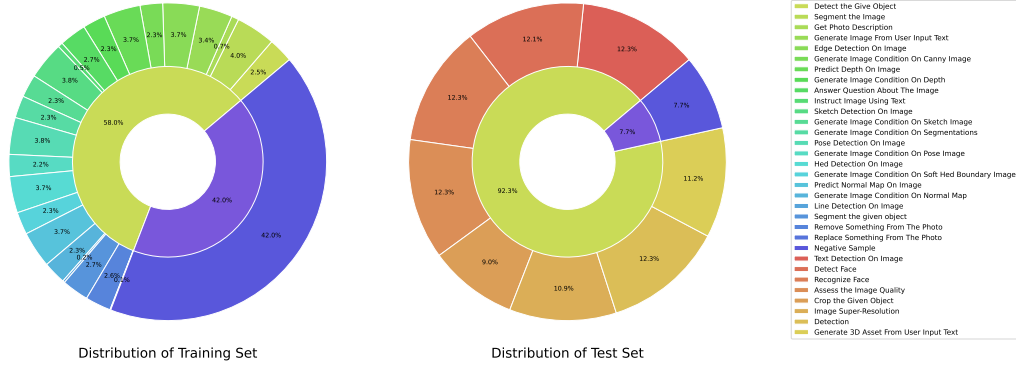


Figure 7: Data distribution of GPT4Tools. The purple piece refers to negative samples, while the others are positive samples.

A.1 Training Set

The training set of GPT4Tools has 71.4K instruction-following data, which includes 35.7K items using tools. Note that these instruction-response pairs are generated from 41K items in Y_S^+ since some actions require two tools. The instructional data in the training set involves 23 tools whose names are shown in Table 5 (marked in gray). The distribution of these 23 tools is illustrated on the left of Figure 7. We employ this training set to instruct the language model to invoke tools.

A.2 Evaluation Set.

The evaluation set consists of two parts: validation set and test set.

Validation. The validation set has 1170 samples in total, which includes the same tools as the training set. The number of each tool is almost 50. This set contains some augmented samples as the training set. Thus, it is utilized to verify the effectiveness of the language model in understanding tools after fine-tuning with the training set.

Test. The test set includes 8 tools unseen by the training set. All unseen tool names are marked in black and shown in Table 5, and their detailed definitions are shown in Table 6. The total number of samples is 652, whose distribution is shown on the right of Figure 7. As this set only involves single-turn samples, it is used to evaluate the zero-shot capability of invoking tools by the language model.

B Prompt

Tool Prompt. The proposed GPT4Tools supports 31 tools, including 23 tools defined in Visual ChatGPT [5] and 8 new tools. They are dependent on image generation models (e.g. ControlNet [44], Stable Diffusion [46], InstructPix2Pix [47], and Shape-E [48]), and image understanding models (e.g. SAM [49], BLIP [22], MMDetection [50], MMOCR [51], MMagic [52], Face Recognition², GroundingDINO [53], and others [54–66]). All tool names are summarized in Table 5, where black texts are the newly defined tools. Detailed descriptions of the new tools are illustrated in Table 6, in which the prompt defines the usage scenario of the tool and its arguments.

Generation Prompt. We encouraged the GPT-3.5 (gpt-3.5-turbo) [3] to generate instruction-following data by utilizing the prompt outlined in Table 7. Subsequently, we filtered out noisy instructions, as exemplified in Table 8. Based on the retained data, we performed augmentation following the steps described in § 3.1, resulting in the tool-related dataset.

Tool-Usage Prompt. During replying to the user command, we encouraged the fine-tuned language model to invoke tools by prompt shown in Table 9. In this prompt, the `<image content>` will be

²https://github.com/ageitgey/face_recognition

Table 6: Details of new tools.

No.	Tool Name	Input	Output	Prompt
1	Text Detection On Image	image path	text on the image	Useful when you want to detect the text in the image. The input to this tool should be a string, representing the image_path.
2	Detection	image path	bounding boxes of objects	Useful when you want to detect all objects of the image, but not detect a certain object according to the text. like: detect all the objects in this image, or detect this image. The input to this tool should be a string, representing the image_path.
3	Image Super-Resolution	image path	image path	Useful when you want to enhance the resolution and quality of low-resolution images. like: enhance this image, restore this image. The input to this tool should be a string, representing the image_path.
4	Generate 3D Asset From User Input Text	text	image path	Useful when you want to generate a 3D asset from a user input text and save it to a file. like: generate a 3D asset of an object or something. The input to this tool should be a string, representing the text used to generate the 3D asset.
5	Crop the Given Object	image path, object name	image path	Useful when you want to crop given objects in the picture. The input to this tool should be a comma separated string of two, representing the image_path, the text description of the object to be cropped.
6	Assess the Image Quality	image path	quality score	Useful when you want to give a quality score for the input image. like: assess a quality score for this image, what is the quality score of this image, or can you give a quality for this image. The input to this tool should be a string, representing the image_path.
7	Recognize Face	image path	text	Useful when you only want to recognize faces in the picture. like: recognize who appears in the photo. The input to this tool should be a string, representing the image_path.
8	Detect Face	image path	image path	Useful when you only want to detect out or tag faces in the picture. like: find all the faces that appear in the picture. tag someone in the picture. The input to this tool should be a string, representing the image_path.

Table 7: Generation Prompt. During generation, <caption> will be replaced with the ground-truth caption and bounding boxes. Green words are the desired instructions.

Given an image whose image path is example.png. Image caption: <caption>. The image caption includes detail image description and each object paired with the bounding box ($x1, y1, x2, y2$). For the bounding box, ($x1, y1$) refers to the top left, and ($x2, y2$) refers to the bottom right. $x1$ less than $x2$, and $y1$ less than $y2$.

Below are N visual tools. Each tool is defined as "<tool name>: <usage scenario>, and <arguments>". Please generate 1 visual instructions for each tools, so you need to generate N visual instruction in total.

The generated instructions should follow the format of "<instruction>, [<tool name>, <arguments>]". Each instruction must relate to the caption and can be solved by the tool. You can not revise the "<tool name>", or add any other fake tools that is not defined. You must keep the correct "<arguments>".

Tools:

<tool name>: <usage scenario>, <arguments>

Note that your generated visual instructions should be related to the image caption extremely. Please generate complex and deceptive instructions as much as possible.

Table 8: Cases of noise during the generation. (✗) indicates the noise examples, while (✓) indicates the corrected examples.

<p>Error Format</p> <p>(✗) Segment the young boy swinging the bat [Segment the Given Object, "example.jpg, young boy swinging the bat"] (<i>The instruction is not separated by a comma.</i>)</p> <p>(✓) Segment the young boy swinging the bat, [Segment the Given Object, "example.jpg, young boy swinging the bat"]</p>
<p>Error Arguments</p> <p>(✗) Make the image look like a painting, [Instruct Image Using Text, "painting"]</p> <p>(✓) Make the image look like a painting, [Instruct Image Using Text, "example.png, painting"]</p>
<p>Error Tools</p> <p>(✗) Generate a real image of a cake and pie display from a sketch, [Generate Image Condition On Canny Image, "example.png, sketch of a cake and pie display"]</p> <p>(✓) Generate a real image of a cake and pie display from a sketch, [Generate Image Condition On Sketch Image, "example.png, sketch of a cake and pie display"]</p>

replaced with the predicted image caption if the <user input> requires the image content as the precondition.

C Case Study

Noise During the Generation of Instructions. While ChatGPT [3] or GPT-4 [4] have demonstrated the ability to generate high-quality data [39, 40], there still are some noises in the generated data. For instance, Table 8 shows three kinds of cases with noise, including the sample with error format, the sample with error arguments, and the sample assigned error tools. Therefore, a practical and effective filtering step is necessary when using data generated by large language models.

Bad Cases of GPT-3.5. As shown in Table 10 and 11, the GPT-3.5 [3] invokes the wrong tools to response the user command. Therefore, when using a language model as a controller to build a generalist model, it is advisable to employ our GPT4Tools to enhance the accuracy of language model actions further.

D Experiment Settings

In § 4, we benchmark tool-usage ability of the language model using a self-built dataset. The fine-tuning configuration is recorded in Table 12.

Table 9: Tool-Usage Prompt. During inference, `<image content>` will be replaced with the result from image caption tools, and `<user input>` will be filled with the user command.

GPT4Tools can handle various text and visual tasks, such as answering questions and providing in-depth explanations and discussions. It generates human-like text and uses tools to indirectly understand images. When referring to images, GPT4Tools follows strict file name rules. To complete visual tasks, GPT4Tools uses tools and stays loyal to observation outputs. Users can provide new images to GPT4Tools with a description, but tools must be used for subsequent tasks.

Tools:

`<tool name>: <usage scenario>, <arguments>`

To use a tool, please use the following format:

Thought: Do I need to use a tool? Yes

Action: the action to take, should be one of <tool name list>

Action Input: the input to the action

Observation: the result of the action

When you have a response to say to the Human, or if you do not need to use a tool, you must use the format:

Thought: Do I need to use a tool? No

AI: [your response here]

Follow file name rules and do not fake non-existent file names. Remember to provide the image file name loyally from the last tool observation.

Previous conversation:

Human: Provide an image named . Description: `<image content>`

AI: Received.

New input: `<user input>`

GPT4Tools needs to use tools to observe images, not directly imagine them. Thoughts and observations in the conversation are only visible to GPT4Tools. When answering human questions, repeat important information. Let's think step by step.

Table 10: Incorrect example from GPT-3.5 (text-davinci-003) [3].

Instruction:

GPT4Tools can handle various text and visual tasks, such as answering questions and providing in-depth explanations and discussions. It generates human-like text and uses tools to indirectly understand images. When referring to images, GPT4Tools follows strict file name rules. To complete visual tasks, GPT4Tools uses tools and stays loyal to observation outputs. Users can provide new images to GPT4Tools with a description, but tools must be used for subsequent tasks.

Tools:

> Get Photo Description: useful when you want to know what is inside the photo. receives image_path as input. The input to this tool should be a string, representing the image_path.
 > Answer Question About The Image: useful when you need an answer for a question based on an image. like: what is the background color of the last image, how many cats in this figure, what is in this figure. The input to this tool should be a comma separated string of two, representing the image_path and the question.
 > Detect the Give Object: useful when you only want to detect or find out given objects in the pictureThe input to this tool should be a comma separated string of two, representing the image_path, the text description of the object to be found.
 > Generate Image Condition On Pose Image: useful when you want to generate a new real image from both the user description and a human pose image. like: generate a real image of a human from this human pose image, or generate a new real image of a human from this pose. The input to this tool should be a comma separated string of two, representing the image_path and the user description.
 > Generate Image Condition On Sketch Image: useful when you want to generate a new real image from both the user description and a scribble image or a sketch image. The input to this tool should be a comma separated string of two, representing the image_path and the user description.

To use a tool, please use the following format:

Thought: Do I need to use a tool? Yes

Action: the action to take, should be one of [Get Photo Description, Answer Question About The Image, Detect the Give Object, Generate Image Condition On Pose Image, Generate Image Condition On Sketch Image]

Action Input: the input to the action

Observation: the result of the action

When you have a response to say to the Human, or if you do not need to use a tool, you must use the format:

Thought: Do I need to use a tool? No

AI: [your response here]

Follow file name rules and do not fake non-existent file names. Remember to provide the image file name loyally from the last tool observation.

Previous conversation:

Human: Provide an image named examples/hybowtyx.png. Description: An open bathtub with sliding glass shower door. A bathroom with gray walls and a clear sliding door. A handicap accessible shower and tub with a glass door. A bathtub shower combination with a hand shower and cradle. An enclosed shower with a window and bathtub. Understand the image using tools.

AI: Received.

New input: **What objects are on the sink?**

GPT4Tools needs to use tools to observe images, not directly imagine them. Thoughts and observations in the conversation are only visible to GPT4Tools. When answering human questions, repeat important information. Let's think step by step.

Thought: Do I need to use a tool?

(X) Response:

Thought: Do I need to use a tool? Yes

Action: Get Photo Description

Action Input: examples/hybowtyx.png

Observation: The image contains a bathtub with a sliding glass shower door, a bathroom with gray walls, a clear sliding door, a handicap accessible shower and tub with a glass door, a bathtub shower combination with a hand shower and cradle, and an enclosed shower with a window and bathtub.

(✓) The Ground-truth Response:

Thought: Do I need to use a tool? Yes

Action: Answer Question About The Image

Action Input: examples/hybowtyx.png, what objects are on the sink

Observation: [output_from_vqa_tool]

Table 11: Incorrect example from GPT-3.5 (text-davinci-003) [3].

Instruction:

GPT4Tools can handle various text and visual tasks, such as answering questions and providing in-depth explanations and discussions. It generates human-like text and uses tools to indirectly understand images. When referring to images, GPT4Tools follows strict file name rules. To complete visual tasks, GPT4Tools uses tools and stays loyal to observation outputs. Users can provide new images to GPT4Tools with a description, but tools must be used for subsequent tasks.

Tools:

> Text Detection On Image: useful when you want to detect the text in the image. The input to this tool should be a string, representing the image_path.
 > Remove Something From The Photo: useful when you want to remove and object or something from the photo from its description or location. The input to this tool should be a comma separated string of two, representing the image_path and the object need to be removed.
 > Hed Detection On Image: useful when you want to detect the soft hed boundary of the image. like: detect the soft hed boundary of this image, or hed boundary detection on image, or perform hed boundary detection on this image, or detect soft hed boundary image of this image. The input to this tool should be a string, representing the image_path.
 > Answer Question About The Image: useful when you need an answer for a question based on an image. like: what is the background color of the last image, how many cats in this figure, what is in this figure. The input to this tool should be a comma separated string of two, representing the image_path and the question.

To use a tool, please use the following format:

Thought: Do I need to use a tool? Yes

Action: the action to take, should be one of [Text Detection On Image, Remove Something From The Photo, Hed Detection On Image, Crop Image, Answer Question About The Image]

Action Input: the input to the action

Observation: the result of the action

When you have a response to say to the Human, or if you do not need to use a tool, you must use the format:

Thought: Do I need to use a tool? No

AI: [your response here]

Follow file name rules and do not fake non-existent file names. Remember to provide the image file name loyally from the last tool observation.

Previous conversation:

Provide an image named image/wuspouwe.png. Description: A man riding a snowboard down a snow covered slope. A man is snowboarding down the snowy mountain. A person is snowboarding in a snowy wooded area. A man in a red jacket snowboarding down a hill. A man is snowboarding alone with trees in the background. Understand the image using tools.

AI: Received.

New input: **What is the color of the man's jacket?**

GPT4Tools needs to use tools to observe images, not directly imagine them. Thoughts and observations in the conversation are only visible to GPT4Tools. When answering human questions, repeat important information. Let's think step by step.

Thought: Do I need to use a tool?

(X) Response:

Thought: Do I need to use a tool? Yes

Action: Hed Detection On Image

Action Input: image/wuspouwe.png

Observation: The man's jacket is red.

(✓) The Ground-truth Response:

Thought: Do I need to use a tool? Yes

Action: Answer Question About The Image

Action Input: image/wuspouwe.png, what is the color of the man's jacket?

Observation: [output_from_vqa_tool]

Table 12: Fine-tuning configuration.

Hyper-parameters	Vicuna [12] & LLaMA [37]	OPT [13]
optimizer	AdamW [45]	AdamW [45]
learning rate	3e-4	1.2e-4
warm steps	100	100
weight decay	0.0	0.0
optimizer momentum	$\beta_1, \beta_2=0.9, 0.999$	$\beta_1, \beta_2=0.9, 0.999$
batch size	512	512
epoch	3	3
max length	2048	2048
LoRA [38] attention dimension (r)	16	16
LoRA [38] scaling alpha (α)	16	16
LoRA [38] drop out	0.05	0.05

References

- [1] 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. 1, 4
- [2] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022. 1, 3
- [3] OpenAI. Chatgpt. <https://openai.com/blog/chatgpt/>, 2023. 1, 2, 3, 4, 6, 7, 11, 13, 15, 16
- [4] OpenAI. Gpt-4 technical report, 2023. 1, 3, 4, 13
- [5] 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. 1, 2, 3, 7, 9, 10, 11
- [6] 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. 1, 2, 3
- [7] Mojtaba Komeili, Kurt Shuster, and Jason Weston. Internet-augmented dialogue generation. *arXiv preprint arXiv:2107.07566*, 2021. 2, 3
- [8] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *ICLR*, 2017. 6
- [9] 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. 3
- [10] 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. 2, 3
- [11] 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. 2, 3, 4
- [12] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. <https://lmsys.org/blog/2023-03-30-vicuna/>, 2023. 2, 3, 4, 7, 8, 9, 17
- [13] 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. 2, 4, 7, 17
- [14] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021. 2, 3, 4
- [15] Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. <https://github.com/kingoflolz/mesh-transformer-jax>, 2021. 2, 4
- [16] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 9118–9147. PMLR, 2022. 2, 4
- [17] Zhengyuan Yang, Zhe Gan, Jianfeng Wang, Xiaowei Hu, Yumao Lu, Zicheng Liu, and Lijuan Wang. An empirical study of GPT-3 for few-shot knowledge-based VQA. In *AAAI*, pages 3081–3089. AAAI Press, 2022. 2, 4
- [18] Xizhou Zhu, Jinguo Zhu, Hao Li, Xiaoshi Wu, Hongsheng Li, Xiaohua Wang, and Jifeng Dai. Uni-perceiver: Pre-training unified architecture for generic perception for zero-shot and few-shot tasks. In *CVPR*, pages 16783–16794. IEEE, 2022. 2

- [19] Jiasen Lu, Christopher Clark, Rowan Zellers, Roozbeh Mottaghi, and Aniruddha Kembhavi. Unified-io: A unified model for vision, language, and multi-modal tasks. *arXiv preprint arXiv:2206.08916*, 2022. 2
- [20] Ting Chen, Saurabh Saxena, Lala Li, Tsung-Yi Lin, David J. Fleet, and Geoffrey E. Hinton. A unified sequence interface for vision tasks. In *NeurIPS*, 2022. 2
- [21] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022. 2
- [22] Junnan Li, Dongxu Li, Caiming Xiong, and Steven C. H. Hoi. BLIP: bootstrapping language-image pre-training for unified vision-language understanding and generation. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 12888–12900. PMLR, 2022. 2, 11
- [23] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. *arXiv preprint arXiv:2301.12597*, 2023.
- [24] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob L. Menick, Sebastian Borgeaud, Andy Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karén Simonyan. Flamingo: a visual language model for few-shot learning. In *NeurIPS*, 2022.
- [25] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023. 2
- [26] 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. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, pages 23318–23340. PMLR, 2022. 2
- [27] Alexander Kolesnikov, André Susano Pinto, Lucas Beyer, Xiaohua Zhai, Jeremiah Harmsen, and Neil Houlsby. Uvim: A unified modeling approach for vision with learned guiding codes. In *NeurIPS*, 2022. 2
- [28] Ruiqi Zhong, Kristy Lee, Zheng Zhang, and Dan Klein. Adapting language models for zero-shot learning by meta-tuning on dataset and prompt collections. In *EMNLP (Findings)*, pages 2856–2878, 2021. 3
- [29] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. In *ICLR*. OpenReview.net, 2022. 3
- [30] Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *arXiv preprint arXiv:2210.11416*, 2022. 3
- [31] Stephen H. Bach, Victor Sanh, Zheng Xin Yong, Albert Webson, Colin Raffel, Nihal V. Nayak, Abheesh Sharma, Taewoon Kim, M. Saiful Bari, Thibault Févry, Zaid Alyafeai, Manan Dey, Andrea Santilli, Zhiqing Sun, Srulik Ben-David, Canwen Xu, Gunjan Chhablani, Han Wang, Jason Alan Fries, Maged Saeed AlShaibani, Shanya Sharma, Urmish Thakker, Khalid Almubarak, Xiangru Tang, Dragomir R. Radev, Mike Tian-Jian Jiang, and Alexander M. Rush. Promptsources: An integrated development environment and repository for natural language prompts. In *ACL*, pages 93–104. Association for Computational Linguistics, 2022.
- [32] Srinivasan Iyer, Xi Victoria Lin, Ramakanth Pasunuru, Todor Mihaylov, Dániel Simig, Ping Yu, Kurt Shuster, Tianlu Wang, Qing Liu, Punit Singh Koura, et al. Opt-impl: Scaling language model instruction meta learning through the lens of generalization. *arXiv preprint arXiv:2212.12017*, 2022. 3
- [33] 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. 3
- [34] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023. 3, 4

- [35] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*, 2023.
- [36] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning, 2023. 3, 4, 9, 10
- [37] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 3, 7, 17
- [38] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *ICLR*, 2022. 3, 6, 7, 17
- [39] Fabrizio Gilardi, Meysam Alizadeh, and Maël Kubli. Chatgpt outperforms crowd-workers for text-annotation tasks. *arXiv preprint arXiv:2303.15056*, 2023. 4, 13
- [40] Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*, 2023. 4, 5, 13
- [41] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *EMNLP*, pages 3980–3990, 2019. 4
- [42] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009. 6
- [43] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In *ECCV*, pages 740–755, 2014. 6
- [44] Lvmin Zhang and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models, 2023. 6, 11
- [45] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 7, 17
- [46] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, pages 10674–10685, 2022. 11
- [47] Tim Brooks, Aleksander Holynski, and Alexei A Efros. Instructpix2pix: Learning to follow image editing instructions. *arXiv preprint arXiv:2211.09800*, 2022. 11
- [48] Heewoo Jun and Alex Nichol. Shap-e: Generating conditional 3d implicit functions. *arXiv preprint arXiv:2305.02463*, 2023. 11
- [49] 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. 11
- [50] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 11
- [51] Zhanghui Kuang, Hongbin Sun, Zhizhong Li, Xiaoyu Yue, Tsui Hin Lin, Jianyong Chen, Huaqiang Wei, Yiqin Zhu, Tong Gao, Wenwei Zhang, Kai Chen, Wayne Zhang, and Dahua Lin. Mmocr: A comprehensive toolbox for text detection, recognition and understanding. *arXiv preprint arXiv:2108.06543*, 2021. 11
- [52] MMLab Contributors. MMLab: OpenMMLab multimodal advanced, generative, and intelligent creation toolbox. <https://github.com/open-mmlab/mmagic>, 2023. 11
- [53] 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. 11
- [54] Jianfeng Wang, Lin Song, Zeming Li, Hongbin Sun, Jian Sun, and Nanning Zheng. End-to-end object detection with fully convolutional network. In *CVPR*, 2021. 11
- [55] Yanwei Li, Lin Song, Yukang Chen, Zeming Li, Xiangyu Zhang, Xingang Wang, and Jian Sun. Learning dynamic routing for semantic segmentation. In *CVPR*, 2020.

- [56] Lin Song, Shiwei Zhang, Gang Yu, and Hongbin Sun. Tacnet: Transition-aware context network for spatio-temporal action detection. In *CVPR*, 2019.
- [57] Lin Song, Yanwei Li, Zeming Li, Gang Yu, Hongbin Sun, Jian Sun, and Nanning Zheng. Learnable tree filter for structure-preserving feature transform. *NIPS*, 2019.
- [58] Shiwei Zhang, Lin Song, Changxin Gao, and Nong Sang. Glnet: Global local network for weakly supervised action localization. *IEEE Transactions on Multimedia*, 22(10):2610–2622, 2019.
- [59] Lin Song, Yanwei Li, Zhengkai Jiang, Zeming Li, Hongbin Sun, Jian Sun, and Nanning Zheng. Fine-grained dynamic head for object detection. *NIPS*, 2020.
- [60] Jianwen Jiang, Yu Cao, Lin Song, Shiwei Zhang, Yunkai Li, Ziyao Xu, Qian Wu, Chuang Gan, Chi Zhang, and Gang Yu. Human centric spatio-temporal action localization. In *ActivityNet Workshop on CVPR*, 2018.
- [61] Lin Song, Yanwei Li, Zhengkai Jiang, Zeming Li, Xiangyu Zhang, Hongbin Sun, Jian Sun, and Nanning Zheng. Rethinking learnable tree filter for generic feature transform. *NIPS*, 2020.
- [62] Lin Song, Songyang Zhang, Songtao Liu, Zeming Li, Xuming He, Hongbin Sun, Jian Sun, and Nanning Zheng. Dynamic grained encoder for vision transformers. *NIPS*, 2021.
- [63] Jinrong Yang, Lin Song, Songtao Liu, Zeming Li, Xiaoping Li, Hongbin Sun, Jian Sun, and Nanning Zheng. Dbq-ssd: Dynamic ball query for efficient 3d object detection. *arXiv preprint arXiv:2207.10909*, 2022.
- [64] Songyang Zhang, Lin Song, Songtao Liu, Zheng Ge, Zeming Li, Xuming He, and Jian Sun. Workshop on autonomous driving at cvpr 2021: Technical report for streaming perception challenge. *arXiv preprint arXiv:2108.04230*, 2021.
- [65] Rui Yang, Lin Song, Yixiao Ge, and Xiu Li. Boxsnake: Polygonal instance segmentation with box supervision. *arXiv preprint arXiv:2303.11630*, 2023.
- [66] Xuchong Zhang, Hongbin Sun, Shiqiang Chen, Lin Song, and Nanning Zheng. Nipm-swmf: Toward efficient fpga design for high-definition large-disparity stereo matching. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(5):1530–1543, 2018. 11