HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in HuggingFace

Yongliang Shen[1*], **Kaitao Song**[2*], **Xu Tan**[2], **Dongsheng Li**[2], **Weiming Lu**[1], **Yueting Zhuang**[1]

Zhejiang University[1], Microsoft Research Asia[2]

`{syl, luwm, yzhuang}@zju.edu.cn, {kaitaosong, xuta, dongsli}@microsoft.com`

Abstract

Solving complicated AI tasks with different domains and modalities is a key step toward artificial general intelligence (AGI). While there are abundant AI models available for different domains and modalities, they cannot handle complicated AI tasks. Considering large language models (LLMs) have exhibited exceptional ability in language understanding, generation, interaction, and reasoning, we advocate that LLMs could act as a controller to manage existing AI models to solve complicated AI tasks and language could be a generic interface to empower this. Based on this philosophy, we present HuggingGPT, a system that leverages LLMs (e.g., ChatGPT) to connect various AI models in machine learning communities (e.g., HuggingFace) to solve AI tasks. Specifically, we use ChatGPT to conduct task planning when receiving a user request, select models according to their function descriptions available in HuggingFace, execute each subtask with the selected AI model, and summarize the response according to the execution results. By leveraging the strong language capability of ChatGPT and abundant AI models in HuggingFace, HuggingGPT is able to cover numerous sophisticated AI tasks in different modalities and domains and achieve impressive results in language, vision, speech, and other challenging tasks, which paves a new way towards AGI [2].

1 Introduction

Large language models (LLMs) [1, 2, 3, 4, 5, 6], such as ChatGPT, have attracted enormous attentions from both academia and industry, due to their remarkable performance on various natural language processing (NLP) tasks. Based on large-scale pre-training on massive text corpora and reinforcement learning from human feedback (RLHF), LLMs can produce superior capability in language understanding, generation, interaction, and reasoning. The powerful capability of LLMs also drives many emergent research topics (*e.g.*, in-context learning [1, 7, 8], instruction learning [9, 10, 11, 12], and chain-of-thought prompting [13, 14, 15, 16]) to further investigate the huge potential of LLMs, and brings unlimited possibilities for us to build artificial general intelligence (AGI) systems.

Despite these great successes, current LLM technologies are still imperfect and confront some urgent challenges on the way to building an AGI system. We discuss them from these aspects: 1) Limited to the input and output forms of text generation, current LLMs lack the ability to process complex information such as vision and speech, regardless of their significant achievements in NLP tasks; 2) In real-world scenarios, some complex tasks are usually composed of multiple sub-tasks, and thus require the scheduling and cooperation of multiple models, which are also beyond the capability of language models; 3) For some challenging tasks, LLMs demonstrate excellent results in zero-shot

---

*The first two authors have equal contributions. This work was done when the first author was an intern at Microsoft Research Asia.

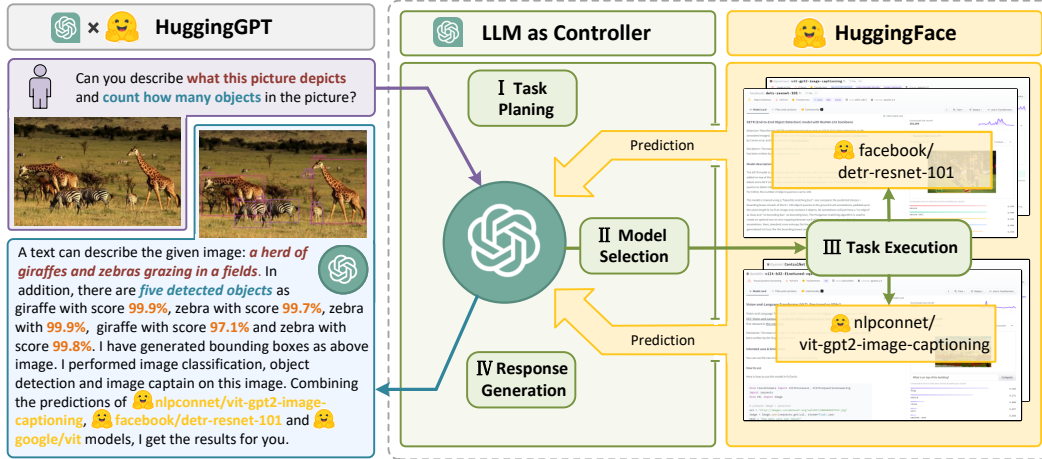[2] `https://github.com/microsoft/JARVIS`

Figure 1: *Language serves as an interface for LLMs (e.g., ChatGPT) to connect numerous AI models (e.g., those in HuggingFace) for solving complicated AI tasks.* In this concept, an LLM acts as a controller, managing and organizing the cooperation of expert models. The LLM first plans a list of tasks based on the user request and then assigns expert models to each task. After the experts execute the tasks, the LLM collects the results and responds to the user.

or few-shot settings, but they are still weaker than some experts (*e.g.*, fine-tuned models). How to address these issues could be the first and also a critical step for LLMs toward AGI systems.

In this paper, we point out that in order to handle complicated AI tasks, LLMs should be able to coordinate with external models to utilize their powers. So, the key point is how to choose suitable middleware to bridge the connections between LLMs and AI models. To address this problem, we notice that each AI model can be denoted as a form of language by summarizing its model function. Therefore, we introduce a concept: "*Language is a generic interface for LLMs to connect AI models*". In other words, by incorporating these model descriptions into prompts, LLMs can be considered as the brain to manage AI models such as planning, scheduling, and cooperation. As a result, this strategy enables LLMs to invoke external models for solving AI tasks. But if we want to integrate multiple AI models into LLMs, another challenge will arise: solving numerous AI tasks needs collecting a large number of high-quality model descriptions, which requires heavy prompt engineering. Coincidentally, we notice that some public ML communities usually offer a wide variety of applicable models with well-defined model descriptions for solving specific AI tasks such as language, vision, and speech. These observations bring us some inspiration: Can we link LLMs (*e.g.*, ChatGPT) with public ML communities (*e.g.*, GitHub, HuggingFace [3], Azure, etc) for solving complex AI tasks via a language-based interface?

Therefore, in this paper, we propose a system called **HuggingGPT** to connect LLMs (*i.e.*, ChatGPT) and ML community (*i.e.*, HuggingFace), which can process inputs from different modalities and solve numerous complex AI tasks. More specifically, for each AI model in HuggingFace, we use its corresponding model description from the library and fuse it into the prompt to establish the connection with ChatGPT. Afterward, in our system, LLMs (*i.e.*, ChatGPT) will act as the brain to determine the answers to the questions of users. Just as shown in Figure 1, the whole process of HuggingGPT can be divided into four stages:

- **Task Planning:** Using ChatGPT to analyze the requests of users to understand their intention, and disassemble them into possible solvable sub-tasks via prompts.
- **Model Selection:** Based on the sub-tasks, ChatGPT will invoke the corresponding models hosted on HuggingFace.
- **Task Execution:** Executing each invoked model and returning the results to ChatGPT.
- **Response Generation:** Finally, using ChatGPT to integrate the prediction of all models, and generate answers for users.

---

[3] https://huggingface.co/models

Benefiting from such a design, HuggingGPT is able to use external models and thus can integrate multimodal perceptual capabilities and handle multiple complex AI tasks. Furthermore, this pipeline also allows our HuggingGPT to continue absorbing the powers from task-specific experts, enabling growable and scalable AI capabilities.

Up to now, our HuggingGPT has integrated hundreds of models on HuggingFace around ChatGPT, covering 24 tasks such as text classification, object detection, semantic segmentation, image generation, question answering, text-to-speech, and text-to-video. Experimental results demonstrate the capabilities of HuggingGPT in processing multimodal information and complicated AI tasks.

In summary, our contributions are as follows:

1. To complement the advantages of large language models and expert models, we propose inter-model cooperation protocols. The large language models act as brains for planning and decision-making, and the small models act as executors for each specific task, providing new ways for designing general AI models.

2. We built HuggingGPT to tackle generalized AI tasks by integrating the HuggingFace hub with 400+ task-specific models around ChatGPT. Through the open collaboration of models, HuggingGPT provides users with multimodal and reliable conversation services.

3. Extensive experiments on multiple challenging AI tasks across language, vision, speech, and cross-modality demonstrate the capability of HuggingGPT in understanding and solving complex tasks from multiple modalities and domains.

2 Related Works

2.1 Large Language Models

In recent years, the field of natural language processing (NLP) has been revolutionized by the emergence of large language models (LLMs)[1, 2, 3, 4, 5, 6], exemplified by models such as GPT-3[1], PaLM [3], and LLaMa [6]. LLMs have demonstrated impressive capabilities in zero-shot and few-shot tasks, as well as more complex tasks such as mathematical problem-solving and commonsense reasoning, due to their massive corpus and intensive training computation.

Some active research areas on LLMs include chain-of-thought prompting (CoT)[13, 14, 15, 16] and instruction tuning[9, 10, 11, 12]. Chain of thought prompting, proposed by [13], prompts the large model to generate problem-solving processes by setting up several case examples, thereby significantly improving the model's reasoning ability. [14] extended this approach to zero-shot chain-of-thought and found that large models can still perform well by using a simple prompt such as "*Let's think step by step*". In contrast, [15] separates program language from natural language, enabling the large language model to generate code logic to solve reasoning problems. Instruction tuning is another approach to large language model applications. Research work such as [10], [11], and [9] have collected and transformed traditional NLP task datasets into instructions, and fine-tuned the large model on the instruction datasets to improve generalization ability on unknown tasks. With instruction tuning, Flan-T5 [12] and Flan-UL2 have outperformed the 650B PaLM [3] model using only 100B parameters. On the other hand, InstrutGPT and ChatGPT employ reinforcement learning from human feedback techniques to align language models with instructions, resulting in exceptional language comprehension and generation abilities.

2.2 Advances in LLM Capabilities

To extend the scope of large language models (LLMs) beyond text generation, contemporary research has investigated two primary approaches. Firstly, some works have devised unified multimodal language models, such as BLIP-2 [17], which utilizes a Q-former to harmonize linguistic and visual semantics, and Kosmos-1 [18], which incorporates visual input into text sequences to amalgamate linguistic and visual inputs. Secondly, other studies have focused on the integration of external tools or models. The pioneering Toolformer [19] introduces external API tags within text sequences, facilitating LLMs' access to external tools. Consequently, numerous works have expanded LLMs to encompass the visual modality. Visual ChatGPT [20] fuses visual foundation models, such as BLIP [21] and ControlNet [22], with LLMs. Visual Programming [23] and ViperGPT [20] apply LLMs to visual objects by employing programming languages, parsing visual queries into

interpretable steps expressed as Python code. Moreover, researchers have endeavored to adapt these LLMs for specialized visual tasks. For instance, Prophet [24] and ChatCaptioner [25] incorporate LLMs into Visual Question Answering and Image Captioning tasks, respectively.

Distinct from these approaches, our proposed HuggingGPT advances towards more general AI capabilities in the following ways: 1) HuggingGPT uses the large language model as an interface to route user requests to expert models, effectively combining the language comprehension capabilities of the large language model with the expertise of other expert models. 2) HuggingGPT is not limited to visual perception tasks but can address tasks in any modality or domain by organizing cooperation among models through the large language model. With the large language model's planning, it is possible to effectively specify task procedures and solve more complex problems. 3) HuggingGPT adopts a more open approach by assigning and organizing tasks based on model descriptions. By providing only the model descriptions, HuggingGPT can continuously and conveniently integrate diverse expert models without altering any structure or prompt settings. This open and continuous manner brings us one step closer to realizing artificial general intelligence (AGI).

3   HuggingGPT

HuggingGPT is a collaborative system that consists of a large language model (LLM) as the controller and numerous expert models as collaborative executors. The workflow of HuggingGPT consists of four stages: task planning, model selection, task execution, and response generation, as shown in Figure 2. 1) An LLM (e.g., ChatGPT) first parses the user request, decomposes it into multiple tasks, and plans the task order and dependency based on its knowledge; 2) The LLM distributes the parsed tasks to expert models according to the model description in HuggingFace; 3) The expert models execute the assigned tasks on the inference endpoints and log the execution information and inference results to LLM; 4) Finally, the LLM summarizes the execution process logs and inference results and returns the summary to the user.

3.1   Task Planning

In the first stage of HuggingGPT, the large language model takes a request from the user and decomposes it into a sequence of structured tasks. Complex requests often involve multiple tasks, and the large language model needs to determine dependencies and execution order for these tasks. To prompt the large language model for effective task planning, HuggingGPT employs both specification-based instruction and demonstration-based parsing in its prompt design. We introduce the details in the following paragraphs.

Specification-based Instruction   The task specification provides a uniform template for tasks and allows the large language model to conduct task parsing through slot filing. HuggingGPT designs four slots for task parsing, which are the task type, task ID, task dependencies, and task arguments:

- Task ID provides a unique identifier for task planning, which is used for references to dependent tasks and their generated resources.

- Task types cover different tasks in language, visual, video, audio, etc. The currently supported task list of HuggingGPT is shown in Tables 1, 2, 3 and 4.

- Task dependencies define the pre-requisite tasks required for execution. The task will be launched only when all the pre-requisite dependent tasks are finished.

- Task arguments contain the list of required arguments for task execution. It contains three subfields populated with text, image, and audio resources according to the task type. They are resolved from either the user's request or the generated resources of the dependent tasks. The corresponding argument types for different task types are shown in Tables 1, 2, 3 and 4.

Benefiting from instruction tuning [9] and reinforcement learning from human feedback [2], the large language model has the ability to follow instructions. HuggingGPT provides these task specifications to the large language model as high-level instructions for analyzing the user's requests and parsing tasks accordingly.
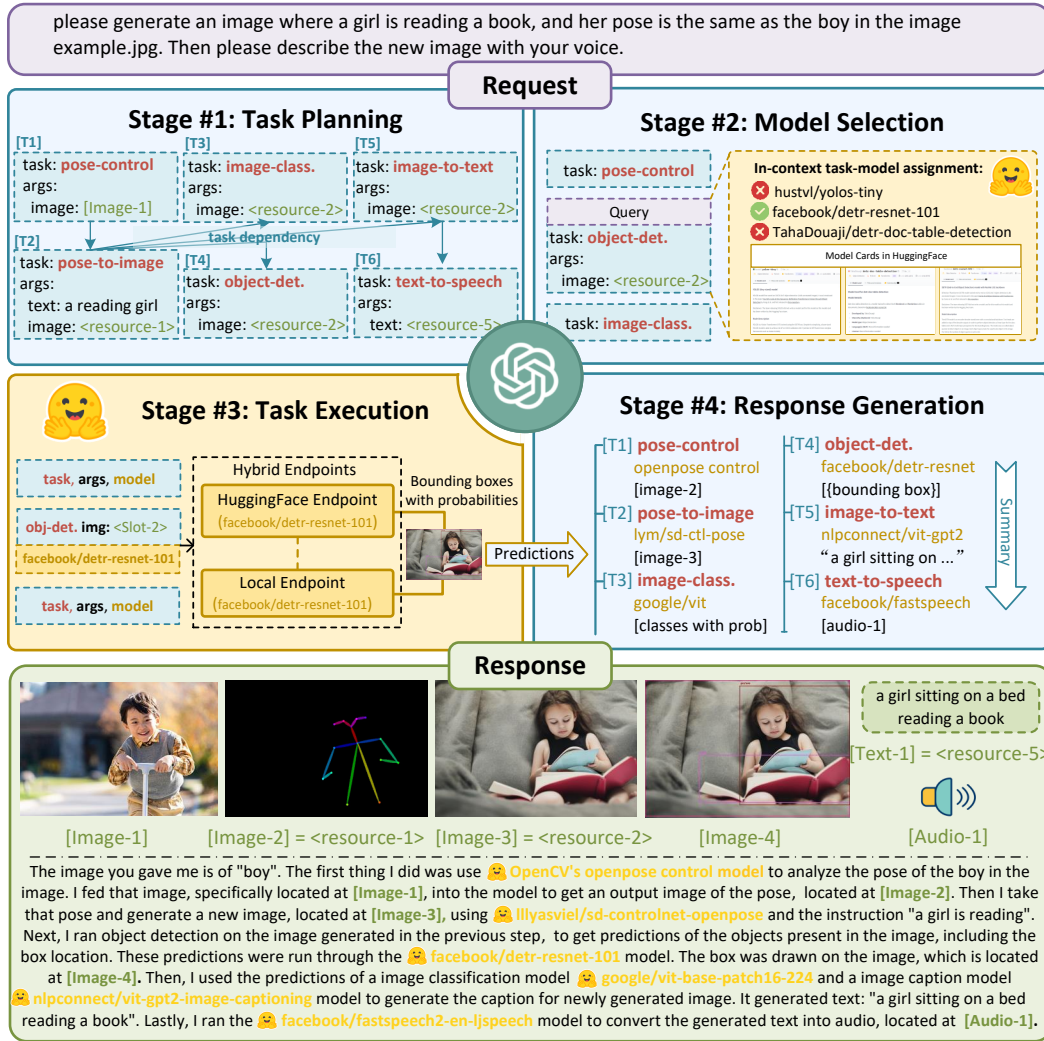
4

Figure 2: Overview of HuggingGPT. With an LLM (e.g., ChatGPT) as the core controller and the expert models as the executors, the workflow of HuggingGPT consists of four stages: 1) **Task planning**: LLM parses user requests into a task list and determines the execution order and resource dependencies among tasks; 2) **Model selection**: LLM assigns appropriate models to tasks based on the description of expert models on HuggingFace; 3) **Task execution**: Expert models on hybrid endpoints execute the assigned tasks based on task order and dependencies; 4) **Response generation**: LLM integrates the inference results of experts and generates a summary of workflow logs to respond to the user.

Task	Args
Text-cls	text
Token-cls	text
Text2text-generation	text
Summarization	text
Translation	text
Question-answering	text
Conversational	text
Text-generation	text
Tabular-cls	text

Table 1: NLP tasks.

Task	Args
Image-to-text	image
Text-to-image	image
VQA	text + image
Segmentation	image
DQA	text + image
Image-cls	image
Image-to-image	image
Object-detection	image
Controlnet-sd	image

Table 2: CV tasks.

Task	Args
Text-to-speech	text
Audio-cls	audio
ASR	audio
Audio-to-audio	audio

Table 3: Audio tasks.

Task	Args
Text-to-video	text
Video-cls	video

Table 4: Video tasks.

Demonstration-based Parsing HuggingGPT introduces in-context learning for more effective task parsing and planning. By injecting several demonstrations into the prompts, HuggingGPT allows the large language model to better understand the intention and criteria for task planning. Each demonstration is a group of input and output on task planning - the user's request and the expected task sequence to be parsed out. Furthermore, these demonstrations, consisting of dependencies between tasks parsed from the user's request, effectively aid HuggingGPT in understanding the logical relationships between tasks and determining the execution order and resource dependency.

Furthermore, the context management of dialogues is essential for chatbots, as it supplies chat logs to facilitate the comprehension of user requests. To incorporate chat context in the task planning stage, we append the following paragraph in the instruction: *The chat log is recorded as {{ Chat Log }}. From the chat log, you can find the history resources for your task planning.*, where *{{ Chat Log }}* are the chat logs between HuggingGPT and the user.

3.2 Model Selection

After parsing the list of tasks, HuggingGPT next needs to match the tasks and models, i.e., select the appropriate model for each task in the task list. For this purpose, we first obtain the descriptions of expert models from the HuggingFace Hub and then dynamically select models for the tasks through the in-context task-model assignment mechanism. This practice allows incremental model access (simply providing the description of the expert models) and is more open and flexible.

Model Descriptions Expert models hosted on the HuggingFace Hub are accompanied by comprehensive model descriptions, which are often provided by the developers. These descriptions contain information on the model's functionality, architecture, supported languages and domains, licensing, and more. This information effectively supports HuggingGPT's decision to select the right model for the task based on the relevance of user requests and model descriptions.

In-Context Task-Model Assignment We approach the assignments of tasks and models as single-choice problems, where potential models are presented as options within a given context. By including the user query and parsed task in the prompt, HuggingGPT can select the most appropriate model for the task at hand. However, due to the constraints regarding maximum context length, it is not always possible to include all relevant model information in the prompt. To address this issue, we filter models based on their task type and only retain those that match the current task type. The remaining models are then ranked based on the number of downloads they have received on Hugging Face, as we believe that this number serves as a reflection of the model's quality to some extent. We then select the top ten models based on this ranking as candidates for HuggingGPT to choose from.

3.3 Task Execution

Once a task is assigned to a specific model, the next step is to execute the task, i.e., to perform model inference. For speedup and computational stability, HuggingGPT runs these models on hybrid inference endpoints. By taking the task arguments as inputs, the models compute the inference results and then send them back to the large language model. To further improve inference efficiency, models that do not have resource dependencies can be parallelized. This means that multiple tasks that have satisfied the prerequisite dependencies can be started simultaneously.

Hybrid Endpoint An ideal scenario is that we only use inference endpoints on HuggingFace. However, in some cases we have to deploy local inference endpoints, such as when inference endpoints for certain models do not exist, the inference is time-consuming, or network access is limited. To keep the system stable and efficient, HuggingGPT pulls and runs some common or time-consuming models locally. The local inference endpoints are fast but cover fewer models, while HuggingFace's inference endpoints are the opposite. Therefore, local endpoints have higher priority than HuggingFace's inference endpoints. Only if the matched model is not deployed locally, HuggingGPT will run the model on the HuggingFace endpoint.

Resource Dependency Despite HuggingGPT's ability to develop the task order through task planning, it can still be challenging to effectively manage resource dependencies between tasks in the task execution stage. The reason for this is that HuggingGPT cannot specify future-generated

6

	Prompt
Task Planning	#1 Task Planning Stage - The AI assistant can parse user input to several tasks: [{"task": task, "id", task_id, "dep": dependency_task_ids, "args": {"text": text, "image": URL, "audio": URL, "video": URL}}]. The "dep" field denotes the id of the previous task which generates a new resource that the current task relies on. A special tag "<resource>-task_id" refers to the generated text image, audio and video in the dependency task with id as task_id. The task MUST be selected from the following options: {{ Available Task List }}. There is a logical relationship between tasks, please note their order. If the user input can't be parsed, you need to reply empty JSON. Here are several cases for your reference: {{ Demonstrations }}. The chat history is recorded as {{ Chat History }}. From this chat history, you can find the path of the user-mentioned resources for your task planning.

Task Planning — Demonstrations

	Demonstrations
Look at /exp1.jpg, Can you tell me how many objects in the picture?	[{"task": "image-to-text", "id": 0, "dep": [-1], "args": {"image": "/exp1.jpg" }}, {"task": "object-detection", "id": 0, "dep": [-1], "args": {"image": "/exp1.jpg" }}]
In /exp2.jpg, what's the animal and what's it doing?	[{"task": "image-to-text", "id": 0, "dep":[-1], "args": {"image": "/exp2.jpg" }}, {"task":"image-classification", "id": 1, "dep": [-1], "args": {"image": "/exp2.jpg" }}, {"task":"object-detection", "id": 2, "dep": [-1], "args": {"image": "/exp2.jpg" }}, {"task": "visual-question-answering", "id": 3, "dep":[-1], "args": {"text": "What's the animal doing?", "image": "/exp2.jpg" }}]
Given an image /exp3.jpg, first generate a hed image, then based on the hed image and a prompt: a girl is reading a book, you need to reply with a new image.	[{"task": "image-to-text", "id": 0, "dep": [-1], "args": {"image": "/examples/boy.jpg" }}, {"task": "openpose-control", "id": 1, "dep": [-1], "args": {"image": "/examples/boy.jpg" }}, {"task": "openpose-text-to-image", "id": 2, "dep": [1], "args": {"text": "a girl is reading a book", "image": "<resource>-1" }}]

	Prompt
Model Selection	#2 Model Selection Stage - Given the user request and the call command, the AI assistant helps the user to select a suitable model from a list of models to process the user request. The AI assistant merely outputs the model id of the most appropriate model. The output must be in a strict JSON format: "id": "id", "reason": "your detail reason for the choice". We have a list of models for you to choose from {{ Candidate Models }}. Please select one model from the list.

	Candidate Models
	[{"model_id": model id #1, "metadata": metadata infos #1, "description": description of model #1}, {"model_id": model id #2, "metadata": metadata infos #2, "description": description of model #2}]

	Prompt
Response Generation	#3 Response Generation Stage - With the input and the inference results, the AI assistant needs to describe the process and results. The previous stages can be formed as - User Input: {{ User Input }}, Task Planning: {{ Tasks }}, Model Selection: {{ Model Assignment }}, Task Execution: {{ Predictions }}. You must first answer the user's request in a straightforward manner. Then describe the task process and show your analysis and model inference results to the user in the first person. If inference results contain a file path, must tell the user the complete file path.

Table 5: The details of the prompt design in HuggingGPT. There are injectable slots in the prompts, such as Demonstrations and Candidate Models. These slots are uniformly replaced with the corresponding text before being fed into the LLM.

resources for tasks during the task planning stage. To address this issue, we use a unique symbol, "<resource>", to manage resource dependencies. Specifically, HuggingGPT identifies the resources generated by the prerequisite task as <resource-task_id>, where task_id is the task id of the prerequisite task. During the task planning stage, if there are tasks that depend on the resource generated by the task with task_id, HuggingGPT sets this symbol to the corresponding resource subfield in the task arguments. Then in the task execution stage, HuggingGPT dynamically substitutes this symbol with the resource generated by the prerequisite task. This strategy empowers HuggingGPT to efficiently handle resource dependencies during task execution.
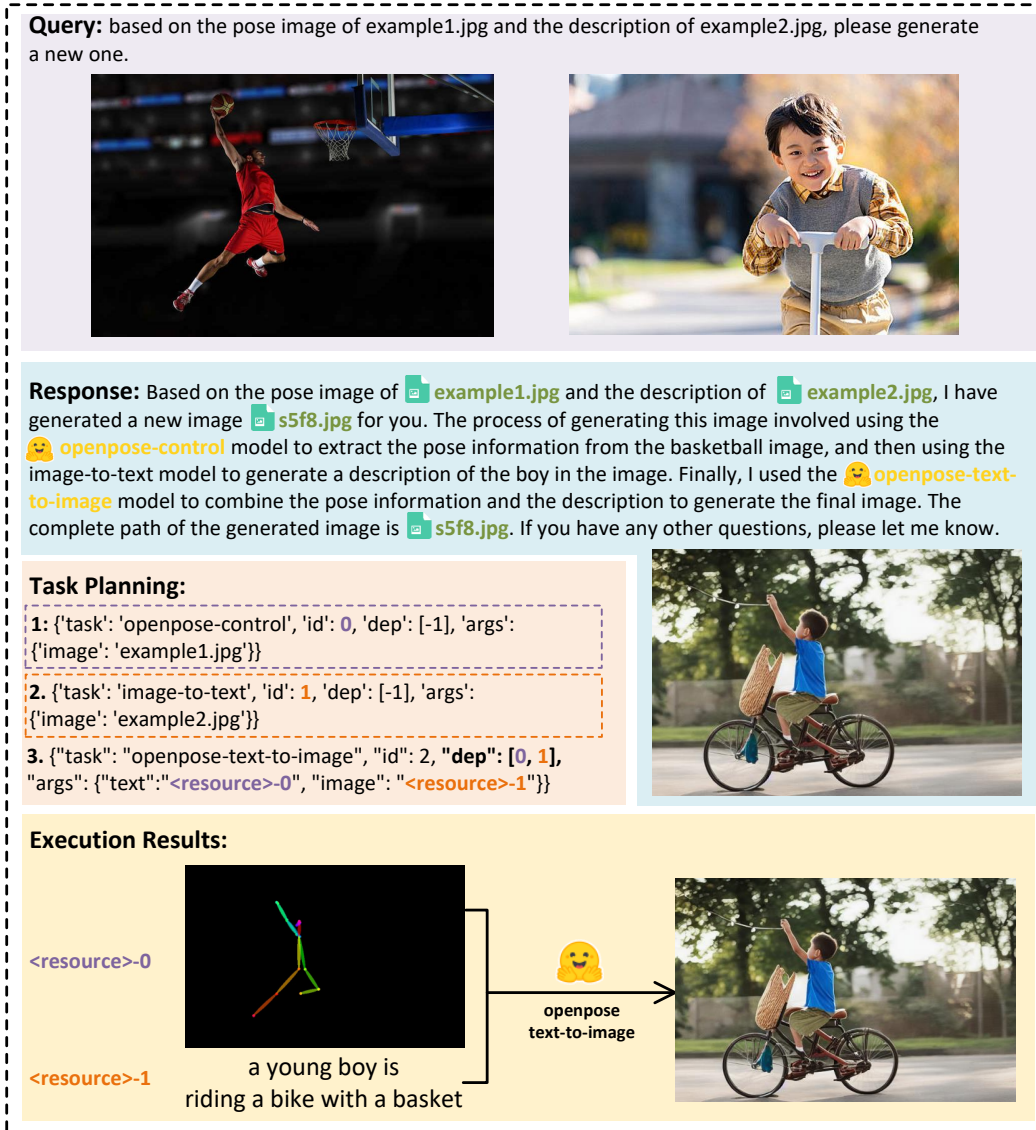
Figure 3: Qualitative analysis of multi-model cooperation with resource dependency.

3.4 Response Generation

After all task executions are completed, HuggingGPT enters the response generation stage. In this stage, HuggingGPT integrates all the information from the previous three stages (task planning, model selection, and task execution) into a concise summary, including the list of planned tasks, the models selected for the tasks, and the inference results of the models. Most important among them are the inference results, which are the supports for HuggingGPT to make the final decisions. These inference results appear in a structured format, such as bounding boxes with detection probabilities in the object detection model, answer distributions in the question-answering model, etc. HuggingGPT allows the LLM to receive these inference results as input and summarize the final responses with confidence levels back to the user.
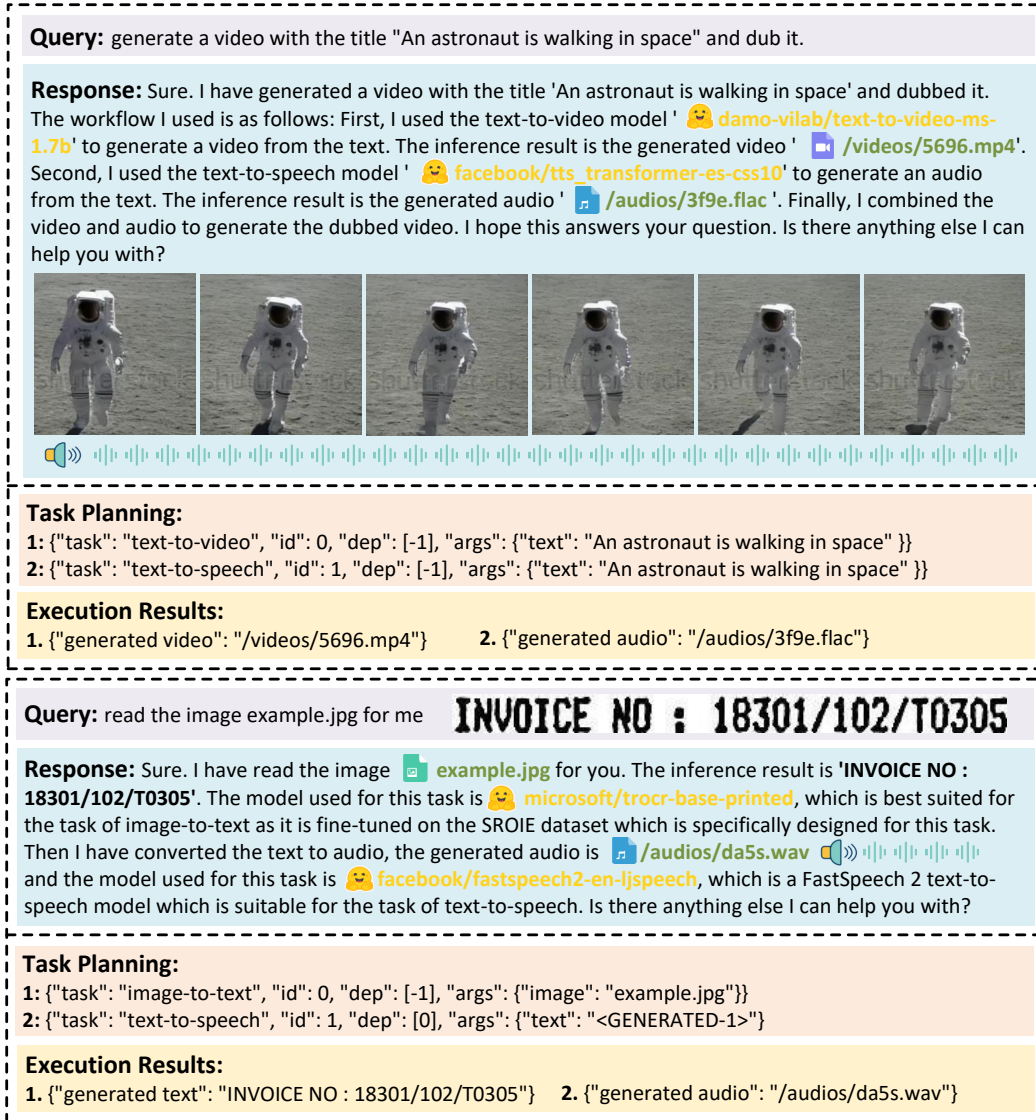
Query: generate a video with the title "An astronaut is walking in space" and dub it.

Response: Sure. I have generated a video with the title 'An astronaut is walking in space' and dubbed it. The workflow I used is as follows: First, I used the text-to-video model ' 🦙 damo-vilab/text-to-video-ms-1.7b' to generate a video from the text. The inference result is the generated video ' 🎥 /videos/5696.mp4'. Second, I used the text-to-speech model ' 🦙 facebook/tts_transformer-es-css10' to generate an audio from the text. The inference result is the generated audio ' 🎵 /audios/3f9e.flac '. Finally, I combined the video and audio to generate the dubbed video. I hope this answers your question. Is there anything else I can help you with?

Task Planning:
1: {"task": "text-to-video", "id": 0, "dep": [-1], "args": {"text": "An astronaut is walking in space" }}
2: {"task": "text-to-speech", "id": 1, "dep": [-1], "args": {"text": "An astronaut is walking in space" }}

Execution Results:
1. {"generated video": "/videos/5696.mp4"}   **2.** {"generated audio": "/audios/3f9e.flac"}

Query: read the image example.jpg for me   **INVOICE NO : 18301/102/T0305**

Response: Sure. I have read the image 🖼 example.jpg for you. The inference result is **'INVOICE NO : 18301/102/T0305'**. The model used for this task is 🦙 microsoft/trocr-base-printed, which is best suited for the task of image-to-text as it is fine-tuned on the SROIE dataset which is specifically designed for this task. Then I have converted the text to audio, the generated audio is 🎵 /audios/da5s.wav 🔊 and the model used for this task is 🦙 facebook/fastspeech2-en-ljspeech, which is a FastSpeech 2 text-to-speech model which is suitable for the task of text-to-speech. Is there anything else I can help you with?

Task Planning:
1: {"task": "image-to-text", "id": 0, "dep": [-1], "args": {"image": "example.jpg"}}
2: {"task": "text-to-speech", "id": 1, "dep": [0], "args": {"text": "<GENERATED-1>"}}

Execution Results:
1. {"generated text": "INVOICE NO : 18301/102/T0305"}   **2.** {"generated audio": "/audios/da5s.wav"}

Figure 4: Qualitative analysis of multi-model cooperation on video and audio modalities.

4 Experiments

4.1 Setting

In our experiments, we employed the `gpt-3.5-turbo` and `text-davinci-003` variants of the GPT models as the large language models, which are publicly accessible through the OpenAI API [4]. To make the LLM output more stable, we set the decoding `temperature` to 0. Also to regulate the LLM output to conform to the expected format, we set the `logit_bias` to 0.1 on the format constraints. We provide detailed prompts designed for the task planning, model selection, and response generation stages in Table 5, where *{{variable}}* indicates that the slot needs to be populated with the corresponding text before the prompt can be fed into the LLM.

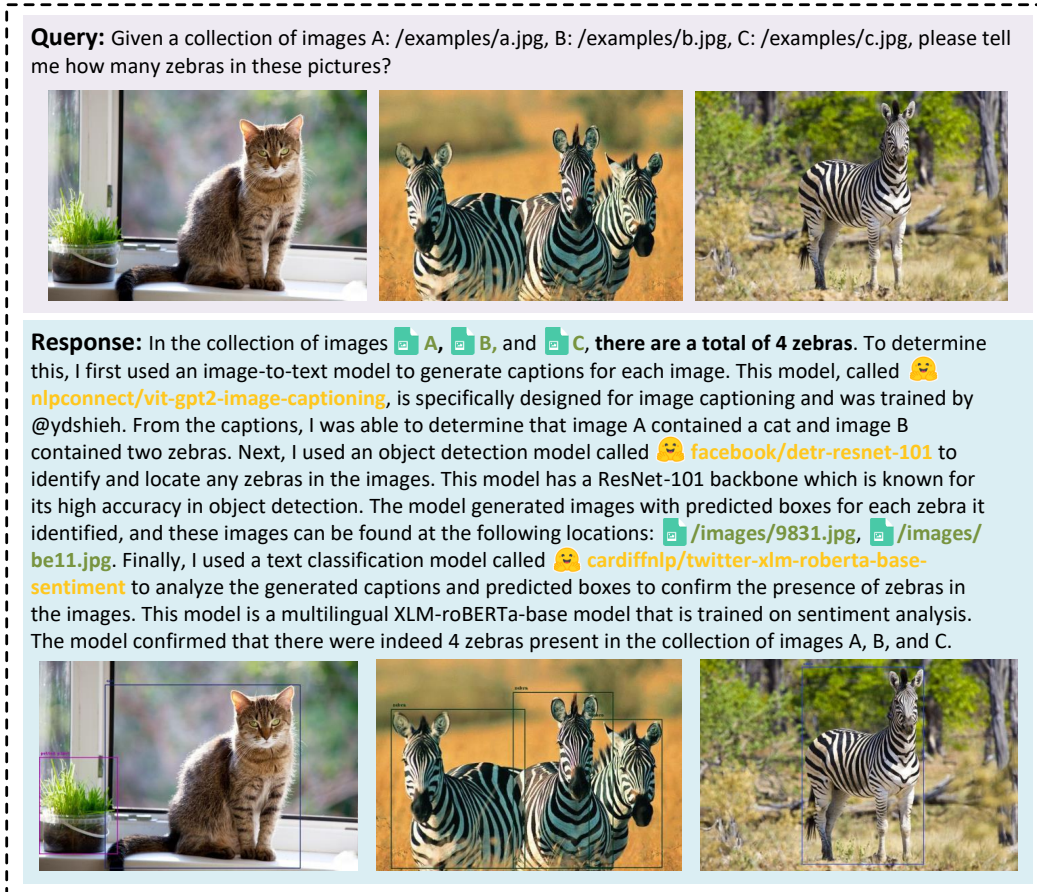[4] https://platform.openai.com/

Figure 5: Qualitative analysis of multi-model cooperation with multiple sources.

4.2 Qualitative Results

In Figures 3, 4 and 5, we show several dialogue demonstrations. In each demo, the user enters a request that may contain multiple tasks or multi-modal resources. Then HuggingGPT relies on the LLM to organize the cooperation of multiple expert models to generate a response back to the user. To make clear the workflow of HuggingGPT, we provide the results of the task planning and task execution stages. Figure 3 shows the process of HuggingGPT working in the presence of resource dependencies between tasks. In this case, HuggingGPT can parse out concrete tasks based on abstract requests from the user, including pose detection, image caption, and pose conditional image generation tasks. Furthermore, HuggingGPT successfully identified the dependencies between task 3 and tasks 1 and 2, and injected the inferred results of tasks 1 and 2 into the input arguments of task 3 after the dependency tasks were completed. Figure 4 demonstrates the conversational ability of HuggingGPT on audio and video modalities. In the two cases, it shows HuggingGPT completes the user-requested text-to-audio and text-to-video tasks via the expert models, respectively. In the top one, the two models are executed in parallel (generating audio and generating video concurrently), and in the bottom one, the two models are executed serially (generating text from the image first, and then generating audio based on the text). This further validates that HuggingGPT can organize the cooperation between models and the resource dependencies between tasks. Figure 5 shows HuggingGPT integrating multiple user-input resources to perform simple reasoning. We can find that HuggingGPT can break up the main task into multiple basic tasks even with multiple resources, and finally integrate the results of multiple inferences from multiple models to get the correct answer.

NLP > Named Entity Recognition

Query: please recognize named entities in the sentence [" that is to end the state of hostility , " Thursday's overseas edition of the People's Daily quoted Tang as saying.]

Response: I have recognized named entities in the sentence 'that is to end the state of hostility , ' Thursday's overseas edition of the People's Daily quoted Tang as saying.'. **The entities recognized are 'People's Daily' and 'Tang'. Both of them are organizations and people respectively.** The model I used is ' 🤗 dslim/bert-base-NER', which is a fine-tuned BERT model that is ready to recognize named entities in the sentence. The inference results are very accurate, with a score of 0.9992 for 'People's Daily' and 0.998 for 'Tang'.

NLP > Text Generation

Query: please complete a sentence [COVID-19 is transmitted via droplets, air-borne,]

Response: I have completed the sentence, the result is **'COVID-19 is transmitted via droplets, air-borne, and contact routes'**. I used a 🤗 microsoft/biogpt model, a domain-specific generative Transformer language model pre-trained on large-scale biomedical literature.

CV > Object Detection

Query: please detect all objects in examples.jpg

Response: I have detected all objects in the image located at 🖼 examples.jpg. I used the 🤗 facebook/detr-resnet-101 model for object detection. It is a DETR (End-to-End Object Detection) model with ResNet-101 backbone, which is known for its high accuracy in object detection tasks. I generated an image with predicted boxes and labels for each object. The complete path of the generated image is examples/d7da.jpg. **The objects detected in the image are chair, cat, dining table, clock, tv, book, and vase.** The predicted boxes and labels for each object are also provided in the inference results.

CV > Visual Question Answering

Query: Look at js4f.jpg, what is the red topping?

Response: Sure, **the red topping on the pizza in the image is tomato.** I used a visual question answering model 🤗 dandelin/vilt-b32-finetuned-vqa to answer the question "What is the red topping?" based on the image. The answer with the highest score is "**tomato**" with a score of 0.9698325395584106.

CV > Depth Estimate

Query: please estimate the depth of ks4j.jpg

Response: I have estimated the depth of the image located at examples/ks4j.jpg using the 🤗 Intel/dpt-large model. This model is specifically trained on 1.4 million images and is a Dense Prediction Transformer (DPT) model. The generated depth image can be found at ' 🖼 j45s.jpg'.

CV > Text-to-Image

Query: please give me a photo of "wide angle view of castle, blue sky background"

Response: Sure, I can do that. For your request, I used the model 🤗 runwayml/stable-diffusion-v1-5 for text-to-image task, because this model is the most suitable for the task as it is a latent text-to-image diffusion model capable of generating high-quality images from text descriptions. For the text-to-image task, I used the 🤗 runwayml/stable-diffusion-v1-5 model to generate an image from the text description you provided. The generated image is located at 🖼 images/asd4.jpg.

Figure 6: Case study on simple tasks (a).

CV > Document Question Answering

Query: look at the document doc.jpg, what is the purchase amount?

Response: Sure. I have used 🤗 **mpira/layoutlm-document-qa** model to analyze the document 📄 **doc.jpg** you provided. After the analysis, **I have found that the purchase amount is $1,000,000,000**. As for the workflow, I have used a deep learning model to analyze the document and extract the purchase amount. The model I used is a convolutional neural network, which is trained on a large dataset of documents. The inference results are the purchase amount of $1,000,000,000. If you need more information, please let me know.

CV > Image to Image

Query: give me a similar image with example.jpg

Sure. For your request, I used the model 🤗 **lambdalabs/sd-image-variations-diffusers** to generate an image from the image you provided. The generated image is located at 📄 **/images/5373.jpg**. If there is nothing in the results, I will tell you that I can't make it. Is there anything else I can help you with?

Video > Generation

Query: please generate a video based on "Spiderman is surfing"

Response: Sure, I can do that. I have generated a video based on your prompt "Spiderman is surfing". The video is saved at the following path: 🎬 **"/videos/4076.mp4"**. 1. Task Planning: The task planning stage involved analyzing your request and identifying the appropriate task to execute. In this case, the task was "text-to-video". 2. Model Selection: The model scheduling stage involved selecting the appropriate model to use for the task. In this case, the only available model was 🤗 **damo-vilab/text-to-video-ms-1.7b**. 3. Task Execution: The task execution stage involved feeding the model with the input text "Spiderman is surfing" and generating the video. The inference result was a generated video saved at the path 🎬 **"/videos/4076.mp4"**. I hope this response meets your requirements.

Audio > Generation

Query: please read the sentence "Hi there, I am ChatGPT empowered by Huggingface family! Yes, I can provide more than one thousand models for dozens of tasks. For more fun and creativity, I have invited Diffusers family to join our team. Feel free to experience it!" for me.

Response: Sure, I can do that. For your request, I used the 🤗 **Facebook/FastSpeech2-en-LJSpeech** model to generate an audio file from the text you provided. The generated audio file is located at ' 🎵 **/audios/d5f3.wav** 🔊 ‖l‖‖l‖‖l‖‖l‖ '. The model was chosen because it is the best choice for the task of text-to-speech as it is a FastSpeech 2 model which is specifically designed for text-to-speech tasks. It is also the most recent model in the list and has the most potential to provide the best results. I hope this answers your question. Is there anything else I can help you with?

Figure 7: Case study on simple tasks (b).

4.3 Case Study on Simple Tasks

HuggingGPT is a multi-model collaborative system that gives LLMs a broader range of capabilities relying on task planning and model selection. We tested HuggingGPT on a wide range of multimodal tasks, and some selected cases are shown in Figures 6 and 7. With the cooperation of a large language model and the expert models, HuggingGPT can solve tasks on a wide range of modalities such as language, image, audio, and video, encompassing a variety of forms of tasks including detection, generation, classification, and question answering. Although these tasks seem simple, the mastery of the basic capabilities of HuggingGPT is a prerequisite for solving complex tasks.

4.4 Case Study on Complex Tasks

User requests may contain multiple implicit tasks or require multi-faceted information, in which case we cannot rely on invoking a single expert model to solve them. To overcome this challenge, HuggingGPT organizes the collaboration of multiple models through task planning. As shown in Figures 8, 9 and 10, we conducted tests to evaluate the effectiveness of HuggingGPT in the case of complex tasks. Figure 8 demonstrates the ability of HuggingGPT to cope with complex tasks in a multi-round conversation scenario. The user splits a complex request into steps and reaches the final

Query #1: Please generate a canny image based on example.jpg

Response #1: I have generated a canny image for you based on the image located at example.jpg. The task process involved selecting the 🤗 **canny-control** model for image processing, as it covers common image processing functions. The inference results include a generated image, which can be found at 🖼️ **images/df8t.jpg**.

Task Planning #1:
1: {"task": "image-to-text", "id": 0, dep: [-1], "args": {"image": "example.jpg"}}
2: {"task": "control-canny", "id": 0, dep: [-1], "args": {"image": "example.jpg"}}

Execution Results #1:
1. {"generated_text": "a large jetliner sitting on top of an airport tarmac "}
2. {"generated_image": "images/df8t.jpg"}

Query #2: then based on the above canny image and a prompt "a photo of a zoo", generate a new image.

Response #2: I have received your request to generate a canny image of a photo of a zoo. I have selected the model 🤗 **lllyasviel/sd-controlnet-canny** as it is the best model for this task. I have processed the image and generated a new image located at 🖼️ **images/ja2f.jpg**. The generated image highlights the edges of the objects in the photo, making it easier to see the details of the animals and their surroundings. Overall, the task was to convert a text prompt 'a photo of a zoo' into a canny image using the selected model.

Task Planning #2:
1: {"task": "image-to-text", "id": 0, dep: [-1], "args": {"image": "example.jpg"}}

Execution Results #2:
1. {"generated_text": "a large jetliner sitting on top of an airport tarmac "}

Query #3: please detect objects in the above image

Response #3:
Sure, here is the result of your command. I hava successfully detected objects in the image you provided. **The detected objects are giraffes and zebras.** I used the 🤗 **facebook/detr-resnet-101** model for object detection, which is specifically designed for object detection tasks and has a ResNet-101 backbone, which is a powerful feature extractor. Additionally, the DETR architecture used in this model is an end-to-end object detection model that has shown promising results in various benchmarks. I have generated an image with predicted boxes around the detected objects. You can find the generated image at the following path: 🖼️ **images/dif2.jpg**. The predicted objects and their corresponding scores are as follows: - Giraffe with a score of 0.999- Zebra with a score of 0.998- Zebra with a score of 0.998- Giraffe with a score of 0.969- Zebra with a score of 0.997. I hope this information is helpful to you.

Task Planning #3:
1: {'task': 'object-detection', 'id': 0, 'dep': [-1], 'args': {'image': 'images/ja2f.png'}}

Execution Results #3:
1. {'generated image with predicted box': '/images/dif2.jpg', 'predicted': [{'box': {'xmax': 760, 'xmin': 515, 'ymax': 437, 'ymin': 136}, 'label': 'giraffe', 'score': 0.9992127418518066}, {'box': {'xmax': 561, 'xmin': 510, 'ymax': 447, 'ymin': 357}, 'label': 'zebra', 'score': 0.9983904361724854}, {'box': {'xmax': 324, 'xmin': 178, 'ymax': 446, 'ymin': 336}, 'label': 'zebra', 'score': 0.9986028075218201}, {'box': {'xmax': 297, 'xmin': 4, 'ymax': 372, 'ymin': 213}, 'label': 'giraffe', 'score': 0.9694724082946777}]}
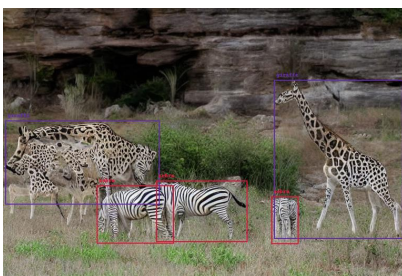
Figure 8: Case study on complex tasks (a).

Query: Describe this image in as much detail as possible. example.jpg

Response: The image you provided appears to show **a family of four dogs playing in the grass**. For this task, I ran multiple CV tasks, including image-to-text, image-classification, object-detection, visual-question-answering, and document-question-answering. I chose specific models for each task that were best suited to addressing your query, based on their training and performance characteristics.

For image-to-text, I chose the 🤗 **nlpconnect/vit-gpt2-image-captioning** model. This model generates an image caption with the given image. I applied this model to the input image and the developed output was: **a family of four dogs are playing in the grass** For image-classification, I selected the 🤗 **google/vit-base-patch16-224** model. This model is trained on natural images dataset and it can predict the label of the image output. I applied this model to the image and get the results showing the risk of each label. It shows the highest risk at "**Rhodesian ridgeback**" with a score of 93.8%
For object-detection, I used 🤗 **facebook/detr-resnet-101 model**. This model is trained on COCO panoptic segmentation for object detection. I used this model to generate an image file with the predicted box. The generated image with predicted box is at: 🖼️ **/images/sd48.jpg**
Regarding your request for image segmentation, I have completed the task using a pre-trained model called 🤗 **facebook/detr-resnet-50-panoptic**. The segmentation mask shows the different regions of the image that belong to different classes. In this case, the model has identified two main classes in the image, which are "**dog**", "**grass**" and "**tree**". The generated image with the segmentation mask can be found at 🖼️ **/images/t4s1.jpg**
For visual question answering, I used 🤗 **dandelin/vilt-b32-finetuned-vqa** model. This model is a Vision-and-Language Transformer (ViLT) model, which is fine-tuned on VQAv2. It is suitable for visual-question-answering tasks, and is likely to provide accurate results for the given image. For the question "what are in the image?" It reply with a prediction **"dogs"** and the confidence is 84.9%.

Task Planning:
1. {"id": "0", "dep": [-1], "task": "image-to-text", "args": {"image": "examples.jpg" }},
2. {"id": "1", "dep": [-1], "task": "image-classification", "args": {"image": "examples.jpg" }},
3. {"id": "2", "dep": [-1], "task": "object-detection", "args": {"image": "examples.jpg" }},
4. {"id": "3", "dep": [-1], "task": "image-segmentation", "args": {"image": "examples.jpg" }},
5. {"id": "4", "dep": [-1], "task": "visual-question-answering", "args": {"text": "describe this image in detail", "image": "examples.jpg" }},

Execution Results:
1. {"generated_text": "a family of four dogs are playing in the grass "}
2. [{'score': 0.938717246055603, 'label': 'Rhodesian ridgeback'}, {'score': 0.01652258075773716, 'label': 'boxer'}, {'score': 0.006381669547408819, 'label': 'Great Dane'}, {'score': 0.006234415341168642, 'label': 'vizsla, Hungarian pointer'}, {'score': 0.005944834090769291, 'label': 'bull mastiff'}]
3. [{"generated image with predicted box": "/images/79f2.jpg", "predicted": [{"box": {"xmax": 463, "xmin": 373, "ymax": 267, "ymin": 199}, "label": "bus", "score": 0.9981155395507812}, {"box": {"xmax": 292, "xmin": 124, "ymax": 224, "ymin": 175}, "label": "airplane", "score": 0.9983609318733215}, {"box": {"xmax": 554, "xmin": 487, "ymax": 277, "ymin": 226}, "label": "person", "score": 0.9910836219787598}]}]
4. {"generated image with segmentation": "/images/3fba.jpg", "predicted": [{"score": 0.989, "label": "grass"}, {"score": 0.999, "label": "dog"}, {"score": 0.999, "label": "tree"},{"score": 0.999, "label": "dog"}]
5. [{'answer': 'dogs', 'score': 0.8488452434539795}, {'answer': 'dog', 'score': 0.04168461635708809}]

Figure 9: Case study on complex tasks (b).

Query: example1.jpg , Can you tell me how many objects in the picture?

Response: There are **8 objects** in the picture. I used the model 🤗 **facebook/detr-resnet-101** for this task which is trained on a popular object detection dataset called COCO, and it has the highest number of downloads. It generated an image with predicted boxes that is available under the path: 🖼 **/images/79f2.jpg** . As for image captioning, I selected the model 🤗 **nlpconnect/vit-gpt2-image-captioning** which is more frequently used and works better in this case. The generated text from the image is '**a large jetliner sitting on top of an airport tarmac.**'

Task Planning:
1: {"task": "object-detection", "id": 0, dep: [-1], "args": {"image": "example1.jpg"}},
2: {"task": "image-to-text", "id": 1, dep: [-1], "args": {"image": "example1.jpg"}}

Execution Results:
1. [{"generated image with predicted box": "/images/79f2.jpg", "predicted": [{"box": {"xmax": 463, "xmin": 373, "ymax": 267, "ymin": 199}, "label": "bus", "score": 0.9981155395507812}, {"box": {"xmax": 292, "xmin": 124, "ymax": 224, "ymin": 175}, "label": "airplane", "score": 0.9983609318733215}, {"box": {"xmax": 554, "xmin": 487, "ymax": 277, "ymin": 226}, "label": "person", "score": 0.9910836219787598}, {"box": {"xmax": 230, "xmin": 219, "ymax": 264, "ymin": 227}, "label": "person", "score": 0.998579740524292}, {"box": {"xmax": 361, "xmin": 330, "ymax": 231, "ymin": 220}, "label": "truck", "score": 0.9066138863563538}, {"box": {"xmax": 639, "xmin": 460, "ymax": 420, "ymin": 142}, "label": "bus", "score": 0.9987366795539856}, {"box": {"xmax": 363, "xmin": 279, "ymax": 222, "ymin": 192}, "label": "airplane", "score": 0.9976897239685059}, {"box": {"xmax": 317, "xmin": 287, "ymax": 230, "ymin": 219}, "label": "truck", "score": 0.9339178800582886}]},
2. {"generated_text": "a large jetliner sitting on top of an airport tarmac "}

Query: example2.jpg , what's the animal and what's it doing?

Response: Sure, **the animal in the image is a tiger and it's laying down.** I performed 3 tasks including image-to-text, image-classification, and visual-question-answering on the image you provided. To accomplish this, I selected models 🤗 **nlpconnect/vit-gpt2-image-captioning**, 🤗 **google/ vit-base-patch16-224** and 🤗 **dandelin/vilt-b32-finetuned-vqa** because they are the perfect models for this use case respectively. According to the results, I have 88% confidence that the animal is a tiger and also have a 77.8% confidence that it's laying down .The generated image with predicted box is stored at the path: 🖼 **/images/d59a.jpg**.

Task Planning:
1: {"task": "image-classification", "id": 0, dep: [-1], "args": {"image": "example2.jpg" }},
2: {"task": "visual-question-answering", "id": 1, dep: [-1], "args": {"text": "What is the animal doing?", "image": "example2.jpg" }}
3: {"task":"image-to-text", "id": 2, dep: [-1], "args": {"image": "example2.jpg" }}

Execution Results:
1. [{'score': 0.8858247399330139, 'label': 'tiger, Panthera tigris'}, {'score': 0.10940514504909515, 'label': 'tiger cat'}, {'score': 0.0006216467591002584, 'label': 'jaguar, panther, Panthera onca, Felis onca'}, {'score': 0.0004262701258994639, 'label': 'dhole, Cuon alpinus'}, {'score': 0.00030842673731967807, 'label': 'lion, king of beasts, Panthera leo'}]
2. {'generated_text': 'a brown and white striped zebra laying on a rock'}
3. [{'score': 0.8858247399330139, 'label': 'tiger, Panthera tigris'}, {'score': 0.10940514504909515, 'label': 'tiger cat'}, {'score': 0.0006216467591002584, 'label': 'jaguar, panther, Panthera onca, Felis onca'}, {'score': 0.0004262701258994639, 'label': 'dhole, Cuon alpinus'}, {'score': 0.00030842673731967807, 'label': 'lion, king of beasts, Panthera leo'}]
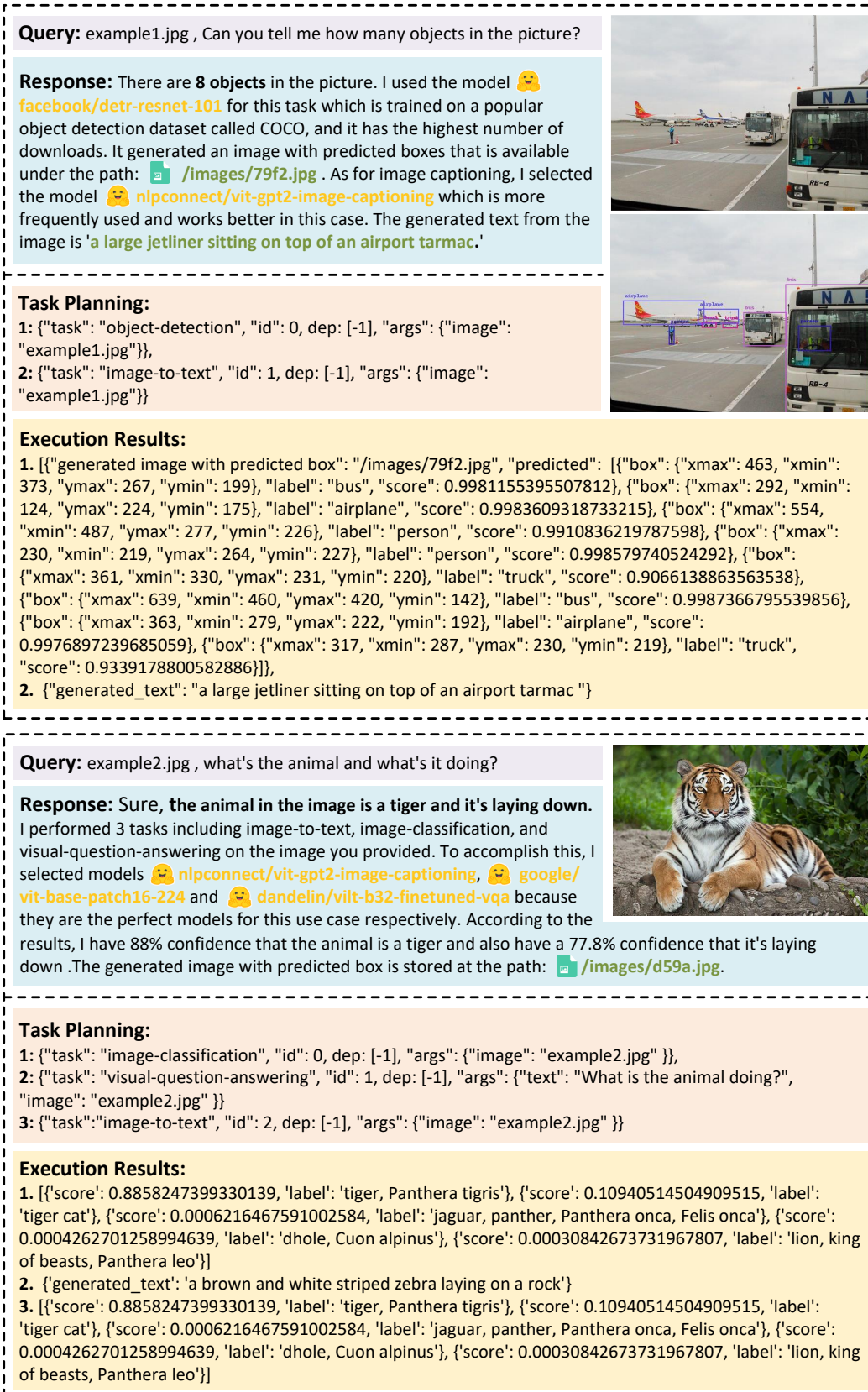
Figure 10: Case study on complex tasks (c).

goal through multiple rounds of requests. We find that HuggingGPT can track the contextual state of user requests through the dialogue context management in the task planning stage, and can address well the resources mentioned by the user as well as task planning. Figure 9 shows that for a "simple" *"describe the image in as much detail as possible"* request, HuggingGPT can expand it into five related tasks, namely image caption, image classification, object detection, segmentation, and visual question answering tasks. HuggingGPT assigns expert models to each task, and these models provide information related to the image from different aspects of the LLM. Finally, the LLM integrates this information to make a comprehensive and detailed description. Figure 10 shows that multiple tasks may be explicitly included in a user request. In these cases, HuggingGPT can accommodate all of the user's requirements, organize multiple expert models to cooperate in parallel, and then let the LLM aggregate the model inference results to respond to the user. In summary, HuggingGPT relies on the collaboration of LLM with external expert models and shows promising performance on various forms of complex tasks.

5 Limitations

HuggingGPT inevitably suffers from some limitations. One of the limitations we are most concerned about is efficiency. The bottleneck of efficiency lies in the inference of the large language model. For each round of user requests, HuggingGPT requires at least one interaction with the large language model during the task planning, model selection, and response generation stages. These interactions greatly increase the response latency and lead to a degradation of user experience. The second is the limitation of the maximum context length. Limited by the maximum number of tokens that the LLM can accept, HuggingGPT also faces a limitation on the maximum context length. We have used the conversation window and only tracked the conversation context in the task planning stage to alleviate it. The third is system stability, which includes two aspects. One is the rebellion that occurs during the inference of large language models. Large language models occasionally fail to conform to instructions when inferring, and the output format may defy expectations, leading to exceptions in the program workflow. The second is the uncontrollable state of the expert model hosted on HuggingFace's inference endpoint. The expert models on HuggingFace may be affected by network latency or service state, leading to errors in the task execution stage.

6 Conclusion

In this paper, we propose a system named HuggingGPT to solve AI tasks, with language as the interface to connect LLMs with AI models. The principle of our system is that an LLM can be viewed as a controller to manage AI models, and can utilize models from ML communities like HuggingFace to solve different requests of users. By exploiting the advantages of LLMs in understanding and reasoning, HuggingGPT can dissect the intent of users and decompose the task into multiple sub-tasks. And then, based on expert model descriptions, HuggingGPT is able to assign the most suitable models for each task and integrate results from different models. By utilizing the ability of numerous AI models from machine learning communities, HuggingGPT demonstrates huge potential in solving challenging AI tasks.

Besides, we also note that the recent rapid development of LLMs has brought a huge impact on academia and industry. We also expect the design of our model can inspire the whole community and pave a new way for LLMs towards AGI.

References

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

[2] Long Ouyang, Jeff 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. 2022.

[3] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, and others. Palm: Scaling language modeling with pathways. *ArXiv*, abs/2204.02311, 2022.

[4] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open Pre-trained Transformer Language Models. *ArXiv*, abs/2205.01068, 2022.

[5] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Zhiyuan Liu, Peng Zhang, Yuxiao Dong, and Jie Tang. Glm-130b: An Open Bilingual Pre-trained Model. *ICLR 2023 poster*, 2023.

[6] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aur'elien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and Efficient Foundation Language Models. *ArXiv*, abs/2302.13971, 2023.

[7] Sang Michael Xie, Aditi Raghunathan, Percy Liang, and Tengyu Ma. An Explanation of In-context Learning as Implicit Bayesian Inference. *ICLR 2022 Poster*, 2022.

[8] Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the Role of Demonstrations: What Makes In-Context Learning Work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2022.

[9] S. Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, and Adam Roberts. The Flan Collection: Designing Data and Methods for Effective Instruction Tuning. *ArXiv*, abs/2301.13688, 2023.

[10] Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, Eshaan Pathak, Giannis Karamanolakis, Haizhi Gary Lai, Ishan Virendrabhai Purohit, Ishani Mondal, Jacob William Anderson, Kirby C. Kuznia, Krima Doshi, Kuntal Kumar Pal, Maitreya Patel, Mehrad Moradshahi, Mihir Parmar, Mirali Purohit, Neeraj Varshney, Phani Rohitha Kaza, Pulkit Verma, Ravsehaj Singh Puri, rushang karia, Savan Doshi, Shailaja Keyur Sampat, Siddhartha Mishra, Sujan Reddy A, Sumanta Patro, Tanay Dixit, Xudong Shen, Chitta Baral, Yejin Choi, Noah A. Smith, Hannaneh Hajishirzi, and Daniel Khashabi. Super-NaturalInstructions: Generalization via Declarative Instructions on 1600+ NLP Tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2022.

[11] S. Iyer, Xiaojuan Lin, Ramakanth Pasunuru, Todor Mihaylov, Daniel Simig, Ping Yu, Kurt Shuster, Tianlu Wang, Qing Liu, Punit Singh Koura, Xian Li, Brian O'Horo, Gabriel Pereyra, Jeff Wang, Christopher Dewan, Asli Celikyilmaz, Luke Zettlemoyer, and Veselin Stoyanov. Opt-IML: Scaling Language Model Instruction Meta Learning through the Lens of Generalization. *ArXiv*, abs/2212.12017, 2022.

[12] Hyung Won Chung, Le Hou, S. Longpre, Barret Zoph, Yi Tay, W. Fedus, Eric Li, Xuezhi Wang, M. Dehghani, Siddhartha Brahma, Albert Webson, S. Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Dasha Valter, Sharan Narang, Gaurav Mishra, A. Yu, Vincent Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, E. Chi, J. Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling Instruction-Finetuned Language Models. *ArXiv*, abs/2210.11416, 2022.

[13] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain of Thought Prompting Elicits Reasoning in Large Language Models. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[14] Takeshi Kojima, Shixiang (Shane) Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large Language Models are Zero-Shot Reasoners. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[15] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided Language Models. *ArXiv*, abs/2211.10435, 2022.

[16] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. *ICLR 2023 poster*, abs/2203.11171, 2023.

[17] Junnan Li, Dongxu Li, S. Savarese, and Steven Hoi. Blip-2: Bootstrapping Language-Image Pre-training with Frozen Image Encoders and Large Language Models. *ArXiv*, abs/2301.12597, 2023.

[18] Shaohan Huang, Li Dong, Wenhui Wang, Y. Hao, Saksham Singhal, Shuming Ma, Tengchao Lv, Lei Cui, O. Mohammed, Qiang Liu, Kriti Aggarwal, Zewen Chi, Johan Bjorck, Vishrav Chaudhary, Subhojit Som, Xia Song, and Furu Wei. Language Is Not All You Need: Aligning Perception with Language Models. *ArXiv*, abs/2302.14045, 2023.

[19] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, M. Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools. *ArXiv*, abs/2302.04761, 2023.

[20] Chenfei Wu, Sheng-Kai Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual ChatGPT: Talking, Drawing and Editing with Visual Foundation Models. *arXiv*, 2023.

[21] 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 *International Conference on Machine Learning (ICML)*, pages 12888–12900, 2022.

[22] Lvmin Zhang and Maneesh Agrawala. Adding Conditional Control to Text-to-Image Diffusion Models. *ArXiv*, abs/2302.05543, 2023.

[23] Tanmay Gupta and Aniruddha Kembhavi. Visual Programming: Compositional visual reasoning without training. *arXiv*, abs/2211.11559, 2022.

[24] Zhenwei Shao, Zhou Yu, Mei Wang, and Jun Yu. Prompting Large Language Models with Answer Heuristics for Knowledge-based Visual Question Answering. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[25] Deyao Zhu, Jun Chen, Kilichbek Haydarov, Xiaoqian Shen, Wenxuan Zhang, and Mohamed Elhoseiny. Chatgpt Asks, BLIP-2 Answers: Automatic Questioning Towards Enriched Visual Descriptions. *arXiv*, 2023.