DORAEMONGPT : TOWARD UNDERSTANDING DYNAMIC SCENES WITH LARGE LANGUAGE MODELS

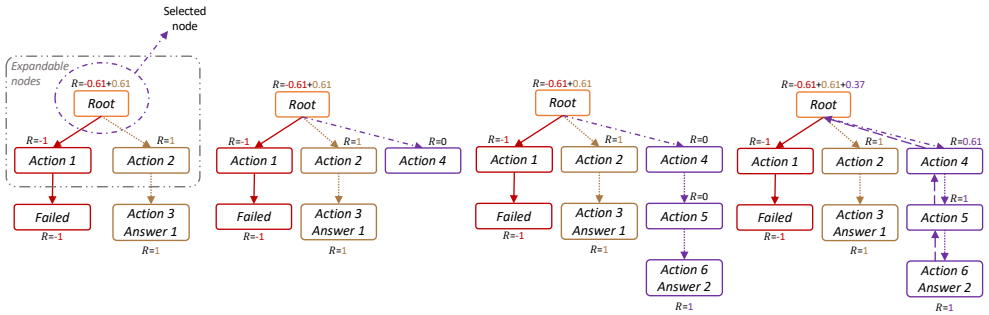
Anonymous authors
Paper under double-blind review

A APPENDIX

This appendix contains additional details for the ICLR 2024 submission, titled *"DoraemonGPT: Toward Solving Real-world Tasks with Large Language Models"*. The appendix is organized as follows:

- §A.1 depicts visual examples regarding the MCTS planner.
- §A.2 offers more implementation details of the MCTS planner.
- §A.3 introduces more in-the-wild examples.
- §A.4 provides inference results on NExT-QA [1] dataset.
- §A.6 analyzes time of inference and efficiency of token usage.
- §A.5 discusses used foundation models.
- §A.8 discusses our limitations.
- §A.9 discusses the broader impacts of our work.

A.1 ILLUSTRATION OF MCTS PLANNER



(a) Node selection (b) Branch Expansion (c) Chain Execution (d) Back-propagation

Figure 1: An illustration of our Monte Carlo Tree Search (MCTS) planner (§A.1). R: the reward of a node. *Root*: the input video and question/task. *Action*: a ReAct [2]-style step in the form of $\langle$*thought, action, action input, observation*$\rangle$.

Fig. 1 illustrates the MCTS planner with one failed solution and two feasible solutions. The illustrated iteration, which produces the second feasible answer, begins with a *node selection* (Fig. 1a), and the *Root* node with the second highest reward is luckily sampled from all expandable non-leaf nodes. Then, the MCTS planner expands the *Root* node with a new child node, *Action 4*, in *Branch Expansion* (Fig. 1b). Following the expansion, the planner continuously executes actions after *Action 4* until getting a new answer, *Answer 2* (Fig. 1c). Lastly, the planner back-propagates the reward of *Answer 2* to its ancestors. Note that those nodes closer to *Answer 2* receive more rewards.

A.2 IMPLEMENTATION DETAILS OF MCTS PLANNER

```
"""
Regarding a given video from {video_filename}, answer the following
    questions as best you can. You have access to the following tools:
{tool_descriptions}
Use the following format:
Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [{tool_names}]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question
Begin!
Question: {input_question}
{ancestor_history}
Thought: {expansion_prompt} {agent_scratchpad}
"""
```

Figure 2: The in-context prompt of the MCTS planner (§A.2).

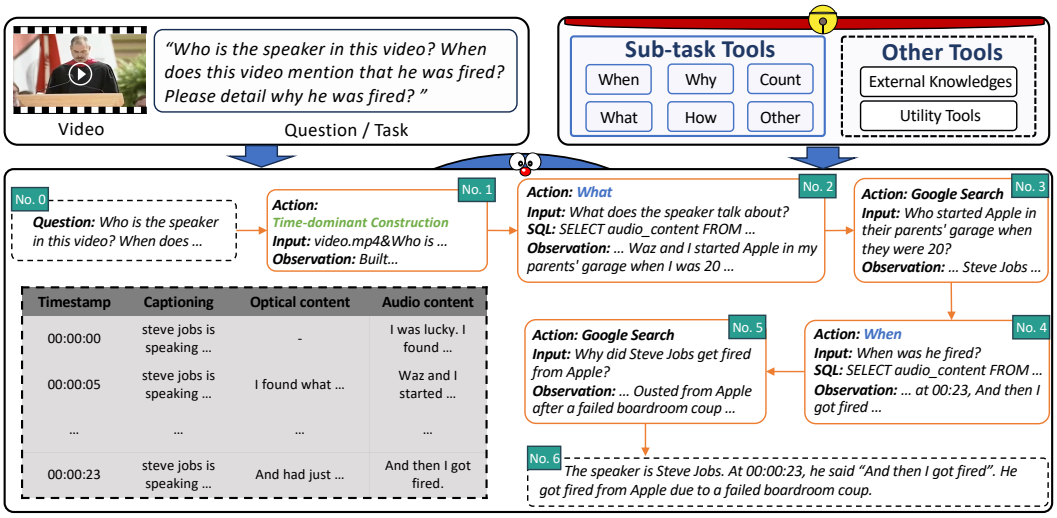
Fig. 2 shows the in-context prompt used in the LLMs of our MCTS planner. By changing the placeholders in the form like {*placeholder*}, the prompt can be adapted to complete **branch expansion** or *chain execution*. The meaning of each placeholder in the prompt is listed below:

- {*video_filename*}: the file path of the input video.
- {*input_question*}: the given question/task regarding the given video.
- {*tool_names*}: the names of tools that can be called by the planner, including sub-task tools, knowledge tools, and utility tools.
- {*tool_descriptions*}: the descriptions of all the callable tools' functions and input format. For example, the description of our *What* sub-task tool is *"Useful when you need to describe the content of a video.....The input to this tool must be a string for the video path and a string for the question. For example: inputs is ./videos/xxx.mp4#What's in the video?"*.
- {*agent_scratchpad*}: the place to put the intermediary output during executing a ReAct [2] step.
- {*ancestor_history*}: the place to put the history of all the ancestor nodes. For example, when selecting a non-root node for *branch expansion*, the action history (which is a string in the form of ⟨*thought, action, action input, observation*⟩ for each node) of all the ancestor nodes of this non-root node will be put in {*ancestor_history*}.
- {*expansion_prompt*}: the place to put the history of all the child nodes for expanding a node, *e.g.*, *"I have thought about the next action before, such as.....I want to think out a different action."*. Only useful in the *branch expansion* phase, set to an empty string for *chain execution*.

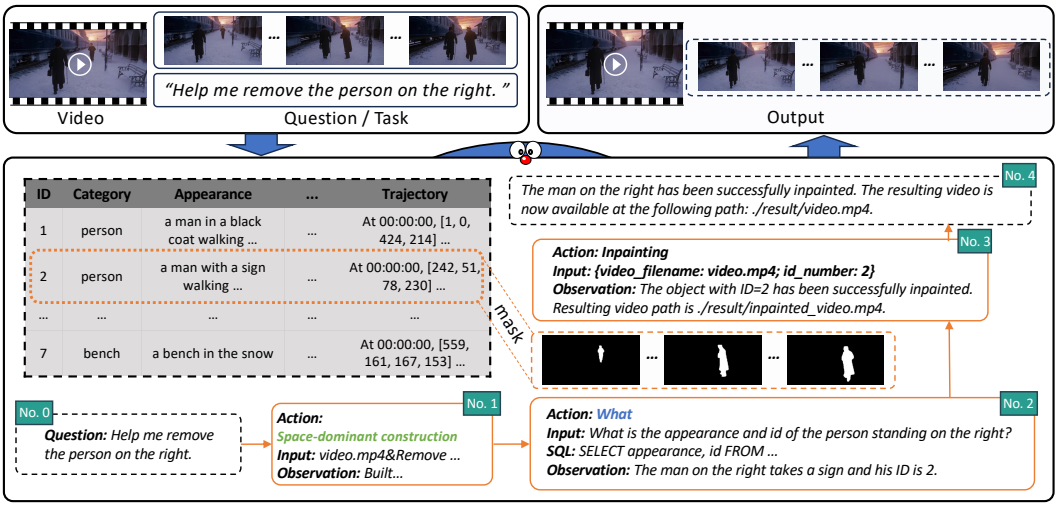
A.3 More In-the-wild Examples

In Fig. 3a, we visualize the reasoning path of a standard video understanding task. As depicted, DoraemonGPT is asked to identify the speaker and analyze information about the dismissal. After several calls to various tools, DoraemonGPT got the right answers. Here we also visualize the *time-dominant* symbolic memory, which is the pivotal part of data processing in DoraemonGPT. Combining it with the well-defined symbolic language (SQL) promises transparency and efficiency.

In addition, we demonstrate an example of video editing by integrating a video inpainting tool. In Fig. 3b, DoraemonGPT is asked to recognize the right person and remove it from the video. To accomplish this, DoraemonGPT constructs the *space-dominant* memory that encompasses the segmentation results for each object within the scene. After recognizing the right person, the inpainting tool is successfully called with an input of the unique ID number assigned to the man on the right, which successfully generates the desired video output.



(a) Video understanding.



(b) Video editing.

Figure 3: In-the-wild examples of DoraemonGPT (§A.3). In the video editing example, the segmentation mask is also visualized.

A.4 Inference Results on NExT-QA

Fig. 4 depicts inference results of DoraemonGPT on NExT-QA [1] dataset. From the top part, we have the following findings: (i) A simple question can be finished within a sub-task tool, *e.g.*, using only the What tool can get the correct answer. (ii) The output of LLM that is not formatted may result in an error case, which is very common in current LLM-driven agents. Similar examples can be observed in the bottom part of the same figure.

As shown in the bottom part of Fig. 4, it's quite possible to pick the wrong tool in the early stages of exploration. Our system is able to explore the planning space with multiple branches further. Interestingly, LLM sometimes considers current information insufficient to make a choice. This is tolerated as our system will eventually vote or summarize all candidate answers.

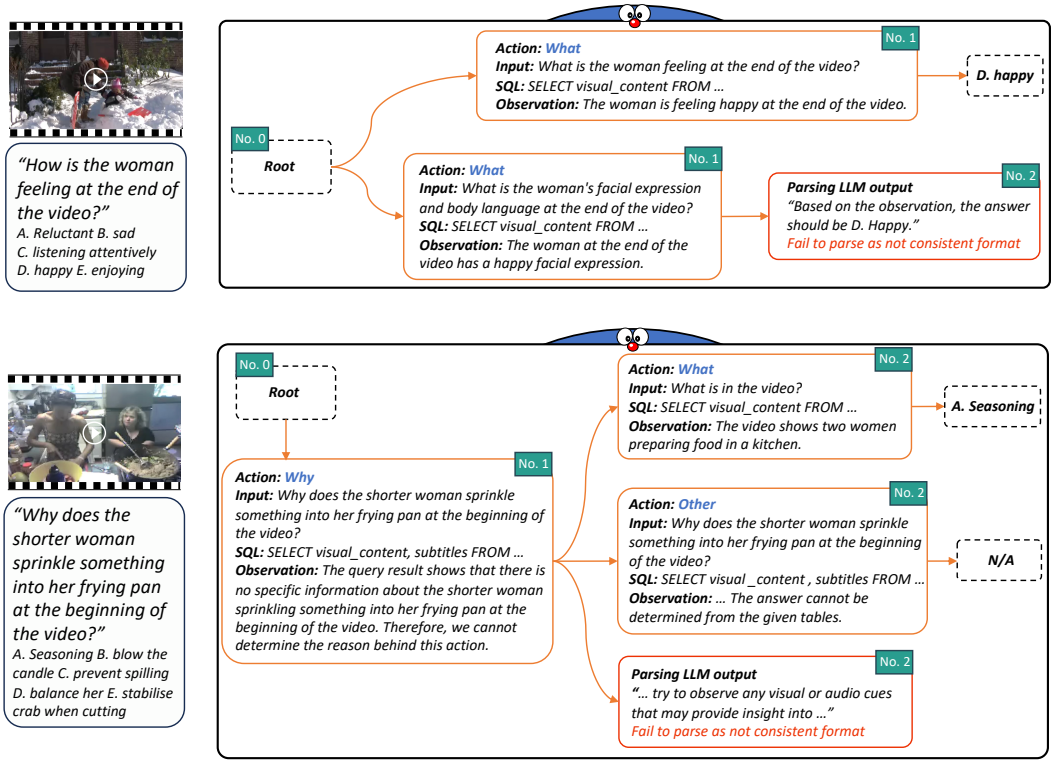


Figure 4: Inference results on NExT-QA [1]. (§A.4)

A.5 Discussion on the Impact of Foundation Models

DoraemonGPT leverages foundation models to extract space-dominant and time-dominant information from videos. Hence, the performance of DoraemonGPT is influenced by the quality of these models as well as its own limitations. This impact can be further summarized as follows:

In space-dominant memory:

Detection (YOLOv8 [3]): The object categories (COCO [4], 80 common categories) are limited by the model, which hinders DoraemonGPT from obtaining information about objects outside these categories. However, YOLOv8 [3] can be replaced with a detection model that supports a wider range of categories (such as one trained on LVIS [5], with 1000+ categories).

Tracking (Deep OC-sort [6]): The current multi-object tracking model is prone to errors in extremely complex scenes (such as those with numerous occluded or similar objects), which affects DoraemonGPT's ability to locate instances in complex videos accurately.

Segmentation (YOLOv8-seg [3]): The segmentation results may not perfectly align with instances' edges, and incomplete segmentation masks can impact the precision of AIGC tools such as video editing (e.g., inpainting).

Appearance description (BLIP [7]/BLIP-2 [8]): The textual descriptions cannot accurately capture all the details of an instance (such as intricate clothing details on a human body), which affects DoraemonGPT's handling of tasks related to detailed descriptions.

Action recognition (InternVideo [9]): The accuracy is limited by the capabilities of the model, which in turn affects DoraemonGPT's ability to handle action-related inquiries.

In time-dominant memory:

Speech recognition (Whisper [10]): Current methods can accurately convert audio to text. However, in multi-party conversation scenarios, the methods still cannot accurately perform voiceprint recognition for multiple speakers and accurately separate the results of different speakers. Addi-

tionally, it is challenging to match multiple voiceprints with the visual IDs of the speakers. This limitation restricts the ability of DoraemonGPT to infer and deduce the identities of speakers in complex multi-party conversation scenarios, relying solely on the inherent capabilities of LLMs.

Optical character recognition (OCR [11]): OCR technology can accurately recognize subtitles and well-structured text. However, it still struggles to robustly handle occluded text and artistic fonts.

Captioning (BLIP [7]/BLIP-2 [8]/InstructBLIP [12]): It cannot guarantee that the textual descriptions can accurately cover all the details in the scene, which can affect DoraemonGPT's ability to handle tasks related to detailed descriptions.

Additionally, the domain of the training set for foundation models also affects DoraemonGPT. For instance, currently, visual foundation models trained on real and common scenarios still struggle with extreme lighting conditions or non-realistic scenes (such as simulations or animations).

A.6 Evaluation on the Inference Time and Token Usage Efficiency

For efficiency comparison, we thoroughly analyze the efficiency of DoraemonGPT in comparison with the baselines, ViperGPT and VideoChat. The tables 1 above provide a detailed analysis of the time required for each foundation model used in memory building. When processing videos at a rate of 1 fps, it takes approximately 1 second (or 0.42/0.47s for space/time-dominant memory) to process a 10s video clip using an NVIDIA-A40 GPU. The actual processing time increases linearly with video length.

Table 1: Token Efficiency (Averaged on the NExT-QA [1] s_val).

Method	Prompt tokens	Node tokens	Steps per Answer	Tokens per Answer	NExT-QA Acc.
ViperGPT [13]	4127	-	-	4127	38.1
VideoChat [14]	722	-	-	**722**	51.0
DoraemonGPT	**617**	34.6	2.3	1498	**54.0**

In comparison, VideoChat creates a time-stamped memory and takes around 2 seconds to process a 10s video at 1 fps. On the other hand, ViperGPT does not construct a memory but generates a code to invoke foundation models. However, there is a 6.7% chance (60 out of 900 videos) that ViperGPT fails to generate an executable code, and it's difficult to fairly compare the average time of calling foundation models in ViperGPT.

Table 2: Time Analysis of Space-Dominant Memory Construction.

Model	BLIP-2 [8]	YOLO-v8 [3]	Deep OC-Sort [6]	InternVideo [9]	Sum
Time(s)	0.09	0.16	0.14	0.03	0.42

Due to the influence of simultaneous requests and network delay on ChatGPT's online server, it's impossible to fairly record the run-time of ChatGPT. Thus, a more equitable efficiency comparison when calling ChatGPT is to record the number of tokens used. As shown in the table above, DoraemonGPT's prompt design is more efficient (617 tokens), which is less than VideoChat's approach of directly incorporating video memory into the prompt (722 tokens) and significantly less than ViperGPT's approach of including a large code definition in the prompt (4127 tokens). Additionally, even though the introduction of our MCTS planner divides the task into multiple nodes/steps, DoraemonGPT still requires far fewer tokens on average to obtain an answer compared to ViperGPT (1498 tokens vs 4127 tokens). Furthermore, DoraemonGPT significantly outperform VideoChat (54.0 vs 51.0) on the challenging NExT-QA dataset.

Table 3: Time Analysis of Time-Dominant Memory Construction.

Model	OCR [11]	Whisper [10]	BLIP-2 [8]	Sum
Time(s)	0.02	0.36	0.09	0.47

A.7 QUANTITAVE RESULT ON TVQA+

Datasets. The TVQA+ [15] dataset is an enhanced version of the original TVQA [16] dataset, augmented with 310.8K bounding boxes to link visual concepts in questions and answers to depicted objects in videos. It's designed for the spatio-temporal video question answering task, which challenges intelligent systems to identify relevant moments and visual concepts to answer natural language questions about videos. For evaluation, we randomly sample 900 samples from the `val` set, resulting in a total of 900 questions (`s_val`).

Evaluation Metric. We report accuracy as in NExT-QA [1].

Table 4: Comparison of our DoraemonGPT with SOTAs on TVQA+ [15]. †: reimplement using the officially released codes. $*$: we filter out those failed executions (*i.e.*, compilation error) of ViperGPT [13] and record the performance on successful executions (802/900 on `s_val`).

Method	Split	Accuracy
†ViperGPT [13]	s_val	26.8
*†ViperGPT [13]	s_val	30.1
†VideoChat [14]	s_val	34.4
DoraemonGPT	s_val	**40.3**

Performance Comparision. The results on the TVQA+ [15] confirms again the superiority of DoraemonGPT. From table 4 we can observe that our approach yields remarkable performance, *i.e.*, DoraemonGPT outperforms ViperGPT [13] and VideoChat [14] by **10.2**% and **5.9**%, respectively. In particular, ViperGPT has a 10.9% probability of generating uncompilable code (98 out of 900 videos). However, even when filtering out these failures, its performance (30.1%) is still lower compared to VideoChat and DoraemonGPT, which are specifically designed for dynamic videos. This is consistent with the findings on NExT-QA [1].

A.8 LIMITATIONS

Despite its comprehensive and conceptually elegant system, DoraemonGPT has some limitations for future studies. First, although TSM is a simple and effective way to decouple and handle spatial-temporal reasoning and DoraemonGPT has shown effectiveness with two task-related memory types (*space-dominant* and *time-dominant*), we believe that by further subdividing the types of tasks, we can introduce more nuanced categories of memory (*e.g.*, human-centric memory) to construct task-related information with greater task-relevance. However, at present, the design of memory types is still a heuristic and manually driven process, lacking an automated design method. Second, the establishment of memory relies on the available foundation models (*e.g.*, BLIP-2 [8]). In other words, foundation models' performance directly influences memory's reliability. Incorrect model predictions will introduce noise into the memory, thereby reducing its reliability and affecting the accuracy of decision-making. Additionally, foundation models may struggle to effectively extract the required video attributes in real-world scenarios that are difficult to generalize (*e.g.*, low light, blurriness, occlusions, *etc.*). Third, the accuracy of planning in DoraemonGPT is limited by the capabilities of LLMs. When using a small-scale or insufficiently trained LLM, the likelihood of DoraemonGPT exploring reasonable solutions may be significantly reduced. Last, while the MCTS planner significantly improves the decision making ability of DoraemonGPT, it also introduces additional computational cost. This means that DoraemonGPT may only be available on high-end computing systems or online LLM services [17], limiting its use in real-time, resource-constrained scenarios.

A.9 BROADER IMPACTS

DoraemonGPT aims to solve real-world dynamic tasks with LLMs and can handle video-based reasoning tasks, potentially revolutionizing several fields. Our system has potential applications in autonomous vehicles, surveillance systems, and interactive robotics, where dynamic understanding and decision making are crucial. However, it is important to consider the ethical implications and potential misuse of such systems. **First**, like many AI systems, DoraemonGPT could be exploited by malicious individuals for video manipulation or generating misleading content, posing threats

to privacy and security. Protecting against such potential misuse requires robust safeguards and measures to detect and prevent malicious activities. **Second**, biases in the training data of LLMs or foundation models could unintentionally perpetuate discriminatory behavior. Mitigating biases and promoting fairness in the training and deployment of DoraemonGPT is essential to ensure equitable outcomes. **Third**, the reliance on external knowledge sources highlights the importance of data access and usage rights. Users and developers must adhere to regulations and ethical guidelines associated with these resources to avoid any legal complications. **Fourth**, the methodology introduced in DoraemonGPT holds potential for application of LLM-driven agents beyond the realm of vision. The rapid expansion of LLM-driven agents opens doors to transformative impacts across various fields [18–24]. DoraemonGPT, with its novel approach to modeling the dynamic aspects of visual scenes, tackles complex tasks through a computer vision lens. This innovation could extend its influence to other domains. For instance, in tool usage, our MCTS planner can offer effective exploration strategies in large solution spaces. Additionally, when it comes to open-world environments, our symbolic memory could provide precise guidances through symbolic language. This is particularly relevant for interactive planning scenarios[18, 20].

REFERENCES

[1] Junbin Xiao, Xindi Shang, Angela Yao, and Tat-Seng Chua. Next-qa: Next phase of question-answering to explaining temporal actions. In *CVPR*, pages 9777–9786, 2021.

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

[3] Glenn Jocher, Ayush Chaurasia, and etc. Yolo by ultralytics. `https://github.com/ultralytics/ultralytics`, 2023.

[4] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[5] Agrim Gupta, Piotr Dollar, and Ross Girshick. Lvis: A dataset for large vocabulary instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5356–5364, 2019.

[6] Gerard Maggiolino, Adnan Ahmad, Jinkun Cao, and Kris Kitani. Deep oc-sort: Multi-pedestrian tracking by adaptive re-identification. *arXiv preprint arXiv:2302.11813*, 2023.

[7] Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *ICML*, pages 12888–12900, 2022.

[8] 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.

[9] Yi Wang, Kunchang Li, Yizhuo Li, Yinan He, Bingkun Huang, Zhiyu Zhao, Hongjie Zhang, Jilan Xu, Yi Liu, Zun Wang, et al. Internvideo: General video foundation models via generative and discriminative learning. *arXiv preprint arXiv:2212.03191*, 2022.

[10] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *ICML*, pages 28492–28518, 2023.

[11] PaddlePaddle. Paddleocr. `https://github.com/PaddlePaddle/PaddleOCR`, 2023.

[12] Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven C. H. Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning. *ArXiv*, 2023.

[13] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *ICCV*, 2023.

[14] KunChang Li, Yinan He, Yi Wang, Yizhuo Li, Wenhai Wang, Ping Luo, Yali Wang, Limin Wang, and Yu Qiao. Videochat: Chat-centric video understanding. *arXiv preprint arXiv:2305.06355*, 2023.

[15] Jie Lei, Licheng Yu, Tamara L Berg, and Mohit Bansal. Tvqa+: Spatio-temporal grounding for video question answering. In *ACL*, 2020.

[16] Jie Lei, Licheng Yu, Mohit Bansal, and Tamara L Berg. Tvqa: Localized, compositional video question answering. In *EMNLP*, 2018.

[17] OpenAI. Introducing chatgpt. *OpenAI Blog*, 2021.

[18] Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. In *NeurIPS*, 2023.

[19] Haoqi Yuan, Chi Zhang, Hongcheng Wang, Feiyang Xie, Penglin Cai, Hao Dong, and Zongqing Lu. Plan4mc: Skill reinforcement learning and planning for open-world minecraft tasks. *arXiv preprint arXiv:2303.16563*, 2023.

[20] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22, 2023.

[21] Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Xingyu Zeng, and Rui Zhao. Tptu: Task planning and tool usage of large language model-based ai agents. *arXiv preprint arXiv:2308.03427*, 2023.

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

[23] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

[24] Zelai Xu, Chao Yu, Fei Fang, Yu Wang, and Yi Wu. Language agents with reinforcement learning for strategic play in the werewolf game. *arXiv preprint arXiv:2310.18940*, 2023.