IEEE-RAS 23rd International Conference on Humanoid Robots (Humanoids), Nancy, France, 2024.

A Comparison of Prompt Engineering Techniques for Task Planning and Execution in Service Robotics

Jonas Bode, Bastian Pätzold, Raphael Memmesheimer, and Sven Behnke

***Abstract*— Recent advances in Large Language Models (LLMs) have been instrumental in autonomous robot control and human-robot interaction by leveraging their vast general knowledge and capabilities to understand and reason across a wide range of tasks and scenarios. Previous works have investigated various prompt engineering techniques for improving the performance of LLMs to accomplish tasks, while others have proposed methods that utilize LLMs to plan and execute tasks based on the available functionalities of a given robot platform. In this work, we consider both lines of research by comparing prompt engineering techniques and combinations thereof within the application of high-level task planning and execution in service robotics. We define a diverse set of tasks and a simple set of functionalities in simulation, and measure task completion accuracy and execution time for several state-of-the-art models. We make our code, including all prompts, available at https://github.com/AIS-Bonn/Prompt_Engineering.**

I. INTRODUCTION

Since the unveiling of OpenAI's ChatGPT in November 2022, the rapid emergence of Large Language Models (LLMs) as a major contributor to human-machine interaction has been rippling through society. With advances in scaling, instruction tuning, and alignment, models like GPT [1], [2], Gemini [3], Llama [4], or Mistral [5] can seemingly understand tasks presented in natural language and respond with answers in natural language that are often appropriate.

In this work, we investigate approaches that leverage these advances for robot control and human-robot interaction [6]–[9], as well as prompt engineering techniques that aim to increase performance across various benchmarks [10]–[12]. In particular, we focus on comparing and combining prompt engineering techniques for the application of task planning and execution in the context of service robotics.

Many applications require autonomous robots to perform tasks in complex, cluttered, dynamic, or unknown environments. In order for such a robot to adapt to new tasks and environments, an expert is often required to understand the task logic and translate it into an implementation based on the robot's capabilities. This severely limits speed and flexibility for robot deployment. LLMs promise to relax this requirement while allowing for intelligent replanning during task execution. To this end, we focus on the challenging field of service robotics [13], [14], requiring robots to interact with non-expert operators in open-ended, highly unstructured environments designed for and shared with humans.

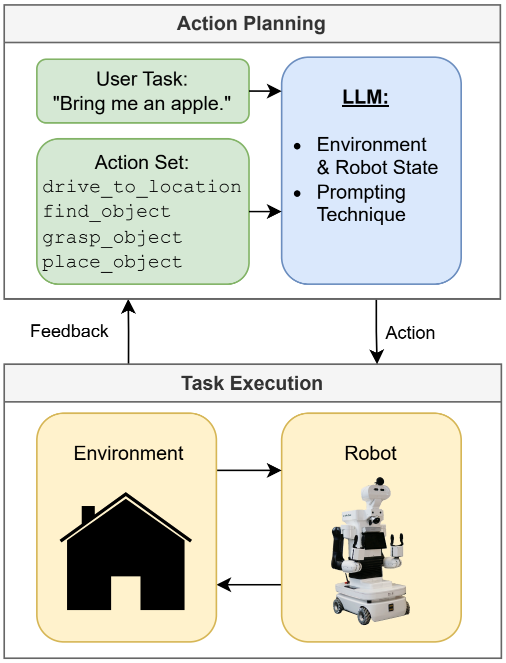

Fig. 1: Overview of our simulated experimental setup to evaluate task completion. The user specifies a task in natural language. The action set describes the robot capabilities. Task planning and action selection are performed by prompting an LLM. The robot-environment simulation executes the action and provides feedback in form of a changed state.

To use an LLM for task planning, one must provide all the necessary information about the environment, the task, and the robot capabilities in textual form. Similarly, for task execution, the implementation of the robot's high-level functionalities must be robust and general enough to account for unknown features of the environment and unexpected events during task execution. In this work, we obviate these problems by employing a simulated environment depicted in Figure 1 to systematically examine the effect of prompt engineering techniques on task completion across several types of tasks and state-of-the-art LLMs.
The contributions of this paper are as follows:

- We derive an LLM-based method allowing zero-shot task execution by sampling relevant action sequences from sets of pre-defined actions received via natural language commands from non-expert operators.
- We thoroughly investigate and evaluate the feasibility and effectiveness of integrating LLMs with various

---

All authors are with the Autonomous Intelligent Systems group, Computer Science Institute VI – Intelligent Systems and Robotics, Lamarr Institute for Machine Learning and Artificial Intelligence, and Center for Robotics, University of Bonn, Germany; `s6jobode@uni-bonn.de`

This work has been funded by the German Ministry of Education and Research under the grant no. 16SV8683, project: Transferzentrum Roboter im Alltag (RimA).

prompt engineering techniques for long-horizon tasks in the domain of service robotics.

II. RELATED WORKS

Using LLMs for high-level robot task planning and for low-level execution control are hot research topics [15], [16].

A. Using LLMs for Robot Task Planning & Execution

In recent years, LLMs have been explored and adopted across a wide range of applications, motivated by harnessing the world knowledge extracted from the vast text data sets they have been trained on.

Vemprala *et al.* [6] are among the proponents for high-level robot task planning using GPT. They outline the design principles of a robotics pipeline that integrates GPT to plan and implement task execution given an objective in textual form. They define a function library that implements various core functionalities for a given robot platform. They then let GPT generate code that implements a task given by a (non-technical) user based on this function library. Finally, the user can provide feedback and corrections to the code before approving it, allowing the robot to execute it. Their pipeline is evaluated in various domains such as manipulation, aerial navigation, and logical reasoning. This approach can generate complex code that incorporates appropriate branching to handle unexpected events during task execution, provided that the library functions support such information. However, the generated code must explicitly anticipate any such cases beforehand, as the method does not allow for dynamic replanning within GPT.

Ichter *et al.* [7] present a similar approach where a set of core functionalities is implemented for a given robot platform and exposed to an LLM input as context. Instead of relying on human feedback, they assign affordance values to all available functions and execute the highest-ranking one. They recalculate the affordance values before each function call to account for the current state of the robot and the environment.

Another LLM-based method for robot task planning is presented by Singh *et al.* [8], combining strengths in common-sense reasoning and code understanding to generate executable plans. Their experiments demonstrate that incorporating programming language features enhances task performance and adapts well to diverse scenarios.

Ding *et al.* [9] propose a method for open-world task planning and situation handling that dynamically integrates commonsense knowledge into a robot's action knowledge, assesses the feasibility of progressing with the existing plan, or determines to adapt the plan accordingly. They integrate their approach on a real robot platform and demonstrate it executing service tasks in a domestic environment.

B. Prompt Engineering Techniques

While LLMs are capable of performing tasks with zero-shot prompting [17], they benefit from in-context examples with few-shot prompting [2]. To further improve performance on tasks requiring complex reasoning, a variety of prompt engineering techniques have been introduced [10]. In this work, we focus on comparing a selection of relevant and applicable prompt engineering techniques suitable for selecting and dynamically adapting sequences of function calls from a predefined library of functions to accomplish a given task.

Wei *et al.* [11] introduced Chain-of-Thought (CoT) prompting, which aims to improve performance on complex tasks by incorporating step-by-step reasoning into the responses provided. For natural language responses, this is easily achieved by modifying the original prompt to elicit step-by-step reasoning in the response. In the context of responses that contain function calls, we can first request step-by-step reasoning in a natural language response with a separate prompt before requesting one or multiple appropriate function calls based on that reasoning.

Yao *et al.* [12] propose ReAct, which facilitates interleaved planning and action generation. Instead of planning ahead and then executing a complex multi-step plan without further reasoning and adapting to intermediate results, they introduce a discrete reasoning step between each action to ground reasoning in the results of past actions and update action plans accordingly.

III. DIALOGUE WITH A SIMULATED ENVIRONMENT

We investigate the effect of prompt engineering techniques across multiple tasks in a simulated environment. In this environment, the robot is provided with pre-defined functions from which an LLM must select and execute the appropriate ones to accomplish a task given in natural language. This process is modeled as a conversation between the robot and the LLM assistant. Once the LLM signals that it has completed the task, we probe the environment to determine if the target condition indicating task success is met.

A. Environment

In order to create an environment that allows an accurate comparison of the effect of prompt engineering techniques, we adopt the setting found at RoboCup@Home [18], [19], which is a major international competition that focuses on domestic service robots.

The arena is designed to replicate a typical domestic dwelling with several rooms: A study room, a parlor, a kitchen, and a bedroom. All tasks start with both robot and operator located in the parlor. Each room contains multiple objects, including common food and household items, with which the robot can interact.

B. Action Set

We use the function-calling feature introduced by OpenAI's API for their GPT line of models [2], which is adopted by other LLMs such as Mistral [5], as well as several open-source inference pipelines. It allows for specifying the signatures of the pre-defined robot functions as a JSON schema. This exposes the function names, parameters, and parameter types, along with the corresponding natural language descriptions for all functions and parameters, to the context of the LLM. The LLM can then refer to these functions when generating natural language reasoning or planning, and it can call any of the functions, including the specification of parameter values, as JSON that can be easily parsed and passed to the simulation. The simulation then adjusts the state of the robot and environment accordingly, and returns a natural language text to the LLM in response to the function call, providing relevant information about the success of the function and its effect on the robot and environment states.

We define the same set of five functions for all tasks:

- `drive_to_location(location_name):` This function moves the robot to the specified location. The available options are the exact names of the four rooms in the simulation: *study*, *parlor*, *kitchen*, and *bedroom*.
- `find_object(object_name_list):` The argument of this function is a list of strings that refer to object names the robot will search for in the room it is in. These object names must exactly match the object names specified for each room in the simulation. For each of the objects, the function returns the number of instances found in the room.
- `grasp_object(object_name):` Once an object has been found, this function lets the robot grasp the specified object. Again, the name of the object must exactly match an object available in the current room. As an additional constraint, the robot can never carry more than two objects at the same time.
- `place_object(object_name):` This function makes the robot place the specified object that is being carried in the current room that the robot is in.
- `exit():` A call to this function signals that the task objective has been accomplished and gracefully terminates the simulation run to the target condition.

If the LLM returns an invalid function call with respect to the specified JSON schema, the simulation run is considered a failure. If the LLM specifies a function parameter that is invalid with respect to the function being unable to execute its intended objective, the function returns a response briefly explaining the problem so that the LLM can adapt and continue the simulation run. Each simulation run is automatically terminated and tested for meeting the target condition after 40 function calls have been executed.

C. Tasks

In our experiments, we define distinct tasks with varying levels of complexity aimed at testing different types of abilities. All tasks are designed to resemble realistic use cases in the sense that they demonstrate a useful application when implemented with a real robot platform, and are related to tasks found in the RoboCup@Home [19] competition. To demonstrate robustness and generalization, we randomly assign the objects and their respective locations for each task. We define the following four tasks:

1) Fetch: The Fetch task requires the robot to pick up an object from another location and to return it to the user. Since Fetch can be solved with a simple sequence of function calls, it mainly tests whether the LLM, in combination with the applied prompt engineering technique, is able to comprehend the scenario, use the function-calling feature, and generate valid JSON appropriate to solve the task. An example of Fetch is: "Please get me a pen from the study".

2) Conditional: The Conditional task extends Fetch by adding the requirement to gather knowledge about the environment during task execution and to adapt its behavior accordingly. In particular, the robot is asked to search for an object and return one of two other objects to the user. The choice of the latter depends on the first object found. This task therefore tests whether the LLM is capable of branching logic. An example of Conditional is: "Check if there is a spoon in the kitchen. If you find one, bring me a pen from the study. If not, bring me a comb from the bedroom".

3) Equals: The Equals task requires the robot to make a numerical observation and repeatedly retrieve an object according to that observation. It therefore tests basic mathematical ability and requires the execution of a logical loop. This results in a long task that requires many function calls, challenging the LLM to maintain focus on accomplishing the task objective. An example of Equals is: "For every orange in the kitchen, move a fork from the kitchen to the parlor".

4) Distribute: In the Distribute task, a given object must be distributed so that every location in the simulation contains at least one instance. The LLM must therefore not only visit all locations, but also make numerous decisions regarding the movement of objects and keep track of their locations. An example of Distribute is: "Please distribute the pens evenly so that each location contains at least one pen. You can start with the pens in the study".

IV. EXPERIMENTS

A. LLM Variants

We focus on evaluating GPT [2] model variants, due to the function-calling feature of the corresponding API, and their top-of-the-line results on benchmarks regarding this feature [20]. In particular, we use GPT-3.5-Turbo-0125 and GPT-4-0125-Preview, referred to as GPT 4 Turbo. We evaluate each condition with respect to tasks and prompt engineering techniques with 50 repetitions for GPT 3.5 Turbo and 20 repetitions for GPT 4 Turbo due to budgetary considerations. For both model variants and all tasks, we use a temperature setting of $\tau = 0$ to elicit factual responses. All other model parameters are set to the default values suggested by OpenAI.

B. Prompt Engineering Techniques

Our simulation procedure allows the robot to execute tasks immediately after providing the LLM with the task description along with the function definitions in an initial prompt. Task completion then proceeds as a dialog of function calls generated by the LLM and corresponding function responses generated by the simulation. This approach serves as a baseline for investigating and comparing the effects of various prompt engineering techniques on task completion and time required. We focus on a selection of popular and

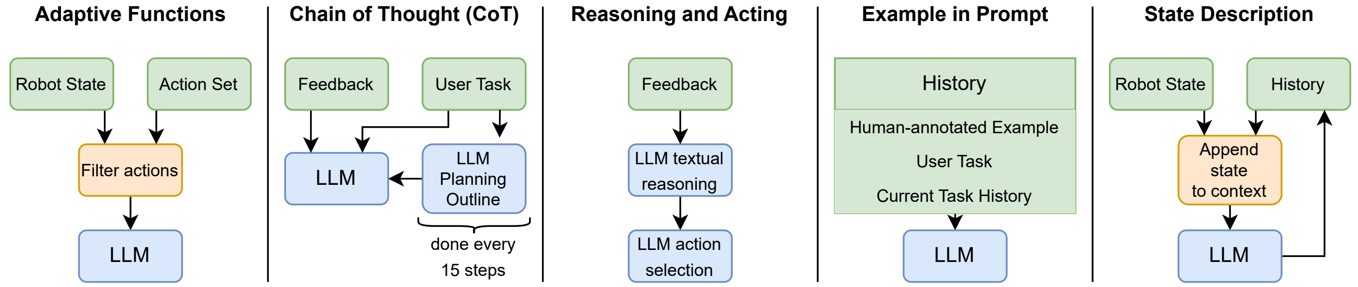

Fig. 2: Control flow diagrams of the five prompt engineering techniques examined. See Sec. IV-B for description.

TABLE I: Comparison of prompt engineering techniques using GPT-3.5-Turbo-0125.

Prompting Technique	**Fetch**		**Conditional**		**Equals**		**Distribute**	
	success rate	mean time [s]	success	time [s]	success	time [s]	success	time [s]
Baseline	0.02	6.55	0.00	9.07	0.00	7.83	0.00	10.47
AF	0.22	3.99	0.00	6.41	0.00	5.23	0.04	11.99
AF + EiP	**1.00**	5.57	0.38	5.43	**0.20**	9.45	0.02	13.79
AF + CoT	0.46	5.66	0.02	5.61	0.00	6.65	0.00	8.25
AF + CoT + EiP	**1.00**	6.76	0.44	7.45	0.16	11.47	0.02	17.46
AF + ReAct + EiP	**1.00**	14.88	0.44	13.76	0.10	22.67	**0.10**	25.86
AF + StD	0.82	4.93	0.00	10.12	0.00	8.96	0.00	9.58
AF + CoT + EiP + StD	**1.00**	8.93	0.36	14.03	0.00	16.40	0.00	20.85
AF + ReAct + EiP + StD	**1.00**	20.69	**0.72**	41.71	0.14	28.37	0.00	68.19

AF – prompting with adaptive functions, EiP – example in prompt, StD – prompt with appended state description. See Sec. IV-B.

applicable prompt engineering techniques for task planning and execution in the service robotics domain. Figure 2 gives an overview of this selection.

We investigate (combinations of) the following five prompt engineering techniques:

1) Adaptive Functions: Based on the robot's state and its knowledge of the environment, we can deduce that certain functions cannot accomplish their intended objective. For example, the robot cannot place an object if it is not currently carrying at least one. Similarly, the robot cannot grasp an object if it is already carrying the specified maximum capacity of two objects. Since the tool-calling feature of the targeted API provides the capability to specify the set of available functions with each prompt, we can exclude such functions accordingly. This approach effectively prevents the LLM from calling those functions. While this is a useful feature for guard-railing the system in general, we intend to use it to reduce the number of failed function calls, and instead focus the LLM's attention on viable functions. We refer to this technique as Adaptive Functions.

2) Chain-of-Thought (CoT): We integrate CoT [11] prompting into our framework by asking the LLM to first provide a textual step-by-step plan detailing how it intends to complete the task given the available functions from the function library. Only after such a plan has been generated, we ask the LLM to call the appropriate functions based on that plan. If the task execution turns out to require many steps, we repeat this planning step every 15 function calls, to ensure that the LLM is always operating according to a plan in case the initial plan is not detailed enough or incomplete.

3) Reasoning and Acting (ReAct): While CoT can be characterized as planning a task in advance, likely involving multiple steps, and then executing the plan without explicitly adapting the original plan or any further reasoning. This may not be sufficient when unexpected or difficult-to-anticipate events occur during task execution. To address this problem, we use ReAct [12] to maintain a tight coupling between task execution and reasoning, by explicitly enforcing the LLM to alternate between the two with dedicated prompts. This also has the advantage of making the decisions of the LLM easily interpretable by looking at the generated reasoning in relation to the generated behavior. This is a useful tool for manually refining function and parameter descriptions.

4) Example in Prompt: Guiding the LLM with a human-annotated example prior to actual task planning and execution, as suggested by Wei *et al.* [11] and Yao *et al.* [12], helps to remove ambiguity regarding the interpretation of task and function descriptions, and is often combined with other prompting techniques. Specifically, we prepend a human-expert generated example of a successful Equals task to the context of the LLM, including the task description and all function calls and responses. The example is designed to be different from any evaluated experimental task.

5) State Description: Finally, we introduce a prompt engineering technique that summarizes the robot's state and knowledge of the environment at the end of the context for before each prompt. Since many tasks involve a long horizon, it can be difficult for the LLM to accurately assess this information when it is required to gather and deduce it from the entire context. These summaries, which we refer to as state descriptions, attempt to mitigate this challenge. To reduce the number of tokens the LLM is prompted with, the context will only ever contain a single state description at the very end of the context. Thus, previous state descriptions are

TABLE II: Comparison of prompt engineering techniques using GPT-4-0125-Preview.

Prompting Technique	Fetch		Conditional		Equals		Distribute	
	success rate	mean time [s]	success	time [s]	success	time [s]	success	time [s]
Baseline	**1.00**	11.57	0.90	14.72	0.85	23.65	0.45	37.38
AF	**1.00**	10.69	0.65	12.06	0.60	22.68	**0.90**	33.64
AF + EiP	**1.00**	11.21	**1.00**	16.03	0.70	21.57	0.65	31.97
AF + CoT	**1.00**	15.14	0.70	18.95	0.70	33.03	0.20	42.32
AF + CoT + EiP	**1.00**	15.07	**1.00**	20.69	0.65	26.26	0.80	36.89
AF + ReAct + EiP	**1.00**	31.05	**1.00**	31.44	0.95	66.45	0.70	64.25
AF + StD	**1.00**	10.45	0.90	12.63	0.65	32.05	**0.90**	29.57
AF + CoT + EiP + StD	**1.00**	15.52	**1.00**	20.63	0.80	28.80	0.75	36.44
AF + ReAct + EiP + StD	**1.00**	29.38	**1.00**	32.86	**1.00**	72.34	0.50	72.25

AF – prompting with adaptive functions, EiP – example in prompt, StD – prompt with appended state description. See Sec. IV-B.

removed from the context with each prompt.

A state description contains information about known objects and their locations, as perceived by the robot. It also details whether the robot is carrying any objects, if so, which ones, and the location of both the robot and the operator.

V. RESULTS

For illustration, Figure 3 shows a sample transcript of a Fetch task using GPT 4 Turbo and CoT prompting. The averaged results of our experiments for all tasks and prompt engineering techniques are shown in Table I for GPT 3.5 Turbo and in Table II for GPT 4 Turbo. For all test conditions, we report the success rate and the average time required to generate a sample. Note that these times only reflect the time spent waiting for LLM responses from the OpenAI API, while the required time of robot actions and simulation feedback can be considered negligible.

On the Fetch task, GPT 4 Turbo achieves a perfect success rate for all prompting techniques. In comparison, GPT 3.5 Turbo requires more complex prompting techniques to achieve a high success rate. The baseline for GPT 3.5 Turbo fails to solve the task in almost all cases. Using adaptive functions, CoT or state descriptions increases the success rate. A perfect success rate for GPT 3.5 Turbo is only achieved by providing an example. Similar results can be observed for the success rate of the Conditional task with GPT 4 Turbo. Again, a perfect success rate is only achieved by providing an example. For the Conditional task with GPT 3.5 Turbo, we observe a significant drop in the success rate when an example is missing. This supports the argument that LLMs are few-shot learners [2]. However, a single example seems to be sufficient to accomplish the task at hand.

We find that more complex prompt engineering techniques do not necessarily lead to higher success rates. On the Equals and Distribute tasks with GPT 3.5 Turbo, the techniques without state descriptions outperform the corresponding technique with state descriptions. However, state descriptions do not always have a negative impact. For the GPT 3.5 Turbo Conditional task, the addition of state descriptions (AF + ReAct + EiP + StD) achieves the highest success rate of 72%. We also observe that using only adaptive functions achieves the best success rate with GPT 4 Turbo for the Distribute task, while achieving the worst success rate for the Conditional and Equals tasks. There is also no clear favorite between CoT and ReAct, as the success rates vary depending on the task, model variant, and other prompt engineering techniques used in conjunction.

Due to the additional prompts used in all prompting techniques besides the adaptive functions, the measured task completion time depends on the technique used. In general, we observe that CoT and even more so ReAct show a longer runtime due to the relatively large number of tokens generated by the LLM for planning and reasoning compared to function calls. Using state descriptions increases the time even more for the same reason. Even though the adaptive functions do not utilize additional prompts, we can see a slight reduction in time compared to the baseline, because the LLM is less likely to call functions that result in failures. In a real application, the response time of an LLM can significantly increase the execution time of a task, although functions that require considerable time themselves provide leeway to mask this problem.

We have successfully used GPT for task planning and execution in several service robotics tasks. We show that the use of appropriate prompt engineering techniques can effectively improve performance. However, overall performance is significantly dependent on the model variant used.

VI. CONCLUSION

In this work, we demonstrated the use of LLMs to understand user commands to plan and execute tasks given an action set and an environment state in the context of service robotics. We evaluated several prompt engineering techniques in a simulated environment, across different tasks and models. GPT 3.5 Turbo performed well on simpler tasks using advanced prompt engineering techniques, but struggled with complex scenarios. GPT 4 Turbo managed to complete complex tasks with mixed reliability, while significantly outperforming GPT 3.5 Turbo on simpler tasks.

We found that while CoT [11] and ReAct [12] improve task completion rates, the best performance is achieved when combined with an example of a successful completion of a related task. This provides an opportunity for human expert knowledge within a task domain or about the robot platform to contribute to the tuning of a robot's behavior.

```
user:
> Please bring me a sponge from the kitchen.
system:
> Briefly summarize the next steps to accomplish
the task in text form.
assistant:
> 1. Drive to the kitchen.
2. Search for a sponge in the kitchen.
3. Grasp the sponge.
4. Drive back to the parlor.
5. Hand over the sponge to the user.
6. Call the exit function to complete the
interaction.
system:
> From now on only use function calls to execute
the task.
assistant:
> drive_to_location: {"location": "kitchen"}
tool:
> You successfully arrived in the new location
kitchen.
assistant:
> find_object: {"object_name_list": ["sponge"]}
tool:
> The following items were found in the kitchen: 3
sponges
assistant:
> grasp_object: {"object_name": "sponge"}
tool:
> You successfully grasped the object sponge.
assistant:
> drive_to_location: {"location": "parlor"}
tool:
> You successfully arrived in the new location
parlor.
assistant:
> place_object: {"object_name": "sponge"}
tool:
> You successfully placed the object sponge.
assistant:
> exit: {}
```

Fig. 3: Transcript showing a Fetch task using adaptive functions and CoT.

Augmenting robotic systems with the reasoning capabilities of LLMs through prompt engineering techniques is a promising area of research. It enables progress toward general-purpose service robots capable of operating in challenging open-ended environments where classical approaches to task planning and execution reach their limits. However, to achieve this goal, work on prompt engineering techniques will remain essential to enable safe and reliable task planning and execution while providing adaptability to environments and robot platforms.

REFERENCES

[1] OpenAI, *ChatGPT*, https://openai.com/blog/chatgpt, Accessed: September 25, 2023.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 1877–1901.

[3] Google AI, *Gemini API*, https://cloud.google.com/vertex-ai/generative-ai/docs/model-reference/gemini, Accessed: March 15, 2024.

[4] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv:2307.09288*, 2023.

[5] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M.-A. Lachaux, P. Stock, *et al.*, "Mistral 7B," *arXiv:2310.06825*, 2023.

[6] S. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor, "ChatGPT for robotics: Design principles and model abilities," Microsoft, Tech. Rep. MSR-TR-2023-8, 2023.

[7] B. Ichter, A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, *et al.*, "Do as I can, not as I say: Grounding language in robotic affordances," in *Conf. on Robot Learning (CoRL)*, vol. 205, 2023, pp. 287–318.

[8] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg, "ProgPrompt: Generating situated robot task plans using large language models," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 11 523–11 530.

[9] Y. Ding, X. Zhang, S. Amiri, N. Cao, H. Yang, A. Kaminski, C. Esselink, and S. Zhang, "Integrating action knowledge and LLMs for task planning and situation handling in open worlds," *Auton. Robots*, vol. 47, no. 8, pp. 981–997, 2023.

[10] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, "A systematic survey of prompt engineering in large language models: Techniques and applications," *arXiv:2402.07927*, 2024.

[11] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, "Chain-of-Thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, 2022, pp. 24 824–24 837.

[12] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in *International Conference on Learning Representations (ICLR)*, 2023.

[13] J. Stückler, M. Schwarz, and S. Behnke, "Mobile manipulation, tool use, and intuitive interaction for cognitive service robot Cosero," *Frontiers Robotics AI*, vol. 3, p. 58, 2016.

[14] R. Memmesheimer, J. Nogga, B. Pätzold, E. Kruzhkov, S. Bultmann, M. Schreiber, J. Bode, B. Karacora, J. Park, A. Savinykh, and S. Behnke, "RoboCup@Home 2024 OPL winner NimbRo: Anthropomorphic service robots using foundation models for perception and planning," in *RoboCup 2024: Robot World Cup XXVII*, to appear, Springer, 2025.

[15] F. Zeng, W. Gan, Y. Wang, *et al.*, "Large language models for robotics: A survey," *arXiv:2311.07226*, 2023.

[16] J. Wang, Z. Wu, Y. Li, H. Jiang, P. Shu, E. Shi, H. Hu, C. Ma, *et al.*, "Large language models for robotics: Opportunities, challenges, and perspectives," *arXiv:2401.04334*, 2024.

[17] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[18] J. Stückler, D. Holz, and S. Behnke, "RoboCup@Home: Demonstrating everyday manipulation skills in RoboCup@Home," *IEEE Robotics and Automation Magazine*, vol. 19, no. 2, pp. 34–42, 2012.

[19] J. Hart, A. Moriarty, K. Pasternak, J. Kummert, A. Hawkin, V. Hassouna, J. D. Pena Narvaez, *et al.*, *RoboCup@Home 2024: Rules and regulations*, https://github.com/RoboCupAtHome/RuleBook/releases/tag/2024.1, 2024.

[20] F. Yan, H. Mao, C. C.-J. Ji, T. Zhang, S. G. Patil, I. Stoica, and J. E. Gonzalez, *Berkeley function calling leaderboard*, https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html, Accessed: March 13, 2024.