OSAgent: Copiloting Operating System with LLM-based Agent

Jiaming Xu, Kaibin Guo, Wuxuan Gong, Runyu Shi
Xiaomi Corporation, China
Email: {xujiaming1, guokaibin, shirunyu}@xiaomi.com

2024 International Joint Conference on Neural Networks (IJCNN) | 979-8-3503-5931-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/IJCNN60899.2024.10650304

***Abstract*—The large language model (LLM) has indeed made significant strides in enhancing human-computer interaction. However, it still faces challenges when dealing with more intricate tasks. Some agents have endeavored to augment LLM capabilities through external memory and tool invocation. Despite these efforts, the task completion rate remains less than satisfactory. This paper introduces a memory-enhanced LLM-based OSAgent designed to assist users in copiloting their operating systems. Specifically, we employ an external memory to store historical user requests, along with corresponding semantic vectors, intent vectors, and human-annotated task planning instructions. When receiving a user request, OSAgent first retrieves similar user requests and task planning instructions as examples to generate prompt, then LLM conducts task planning to manipulate OS tools. Extensive experiments were conducted on mobile operating systems employing seven LLMs. The results are indicative of OSAgent's remarkable proficiency in invoking AI capabilities, third-party applications, system settings, and data resources.**

***Index Terms*—LLM-based Agent, Copiloting Operating System, Task Planning**

I. Introduction

Recently, through the acquisition of vast amounts of web knowledge, large language models (LLMs) [1]–[7] have demonstrated remarkable potential in language understanding, generation, interaction, and reasoning. Since language learning from massive data is multitasking in itself, LLM is considered as a technical approach to achieve AGI. However, the capabilities of LLM are limited to language generation, and it cannot assist human well in complex task.

To address this issue, some efforts have been made to incorporate tasks from multiple modalities, such as GPT-4 [1] and Gato [8]. However, the potential introduced by this direction remains limited, and the associated learning costs are exceedingly high. Another line of work involves the development of LLM-based Agents [9]–[13], including HuggingGPT [14] and AutoGPT [15], which extend LLM capabilities through external memory and the utilization of auxiliary tools. Nevertheless, these approaches exhibit a lower task completion rate. For instance, HuggingGPT can manipulate the model, but it focuses on AI model community and is constrained in terms of model types and only supports three types of input data: text, image, and audio, with an accuracy of merely 46%. AutoGPT's primary contribution lies in its conceptual framework. However, its performance is inconsistent and prone to entering infinite loops, and the range of executable tasks is also limited, primarily encompassing search queries and document organization. Memory-enhancement methods such as Langchain [16] primarily rely on matching relevant documents from a vector knowledge base and incorporating them into the context for guidance.To enhance the capability of LLM, recent works have expanded the length of context up to 100K+, such as Claude, RMT [17], and Microsoft have expanded the length of the context to 1B further [18]. In fact, it is difficult to effectively improve model performance by drastically increasing the context length [19]. Overall, AutoGPT depends more on the performance of GPT-4, while HuggingGPT is limited to the available expert models. Drastically increasing the context length doesn't lead to significantly greater performance.

In contrast, if an Agent is intended to control an operating system, it must contend with more complex scenarios and data, as depicted in Figure 1. This encompasses invoking AI capabilities, third-party applications, system settings, and data resources. The primary challenge we face is how to design a highly precise agent.

In this paper, we point out that for copiloting an operating system, an LLM-based Agent should exhibit a human-like ability to perform task planning by recalling similar scenarios, analogous behaviors, and comparable guidance from a long-term memory. As illustrated in Figure 2, the semantic and intent relationships in distinct user requests exhibit variations in both semantic and intent spaces. Consequently, data retrieved based on content similarity may not always be effective, potentially leading to misguidance. Furthermore, for the OSAgent, the intent domain encompasses AI capabilities, third-party applications, system settings, and data resources. The differences in intent and parameters can result in lengthy descriptions of complete tasks. On one hand, this may easily exceed the length constraints of the LLM. On the other hand, it may provide the LLM with excessive, yet inconsequential, reference information.

In this paper, we propose a Memory-Augmented OSAgent, where the memory includes two vector spaces. Specifically, we augment the semantic encoding module with additional training of an intent encoding module and an intent classification module. Leveraging both semantic and intent encoding, we construct two vector databases within the memory module. When the OSAgent receives a new user request, the retrieval

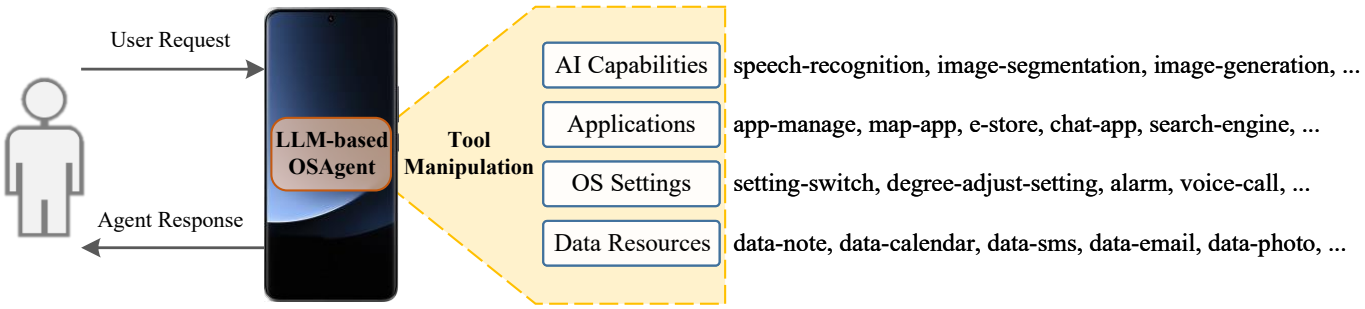
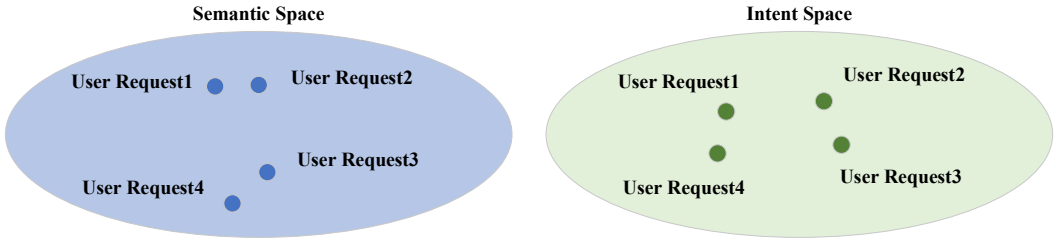


Fig. 1. Copiloting Operating System with LLM-based Agent. In this concept, an LLM acts as an Agent to manipulate AI capabilities, third-party applications, system settings, and data resources on the operating system.



User Request1: I want to watch the movie Ready Player One.
User Request2: I want to search for the plot of the movie Ready Player One.
User Request3: Help me search for biographies related to Oppenheimer.
User Request4: Help me play the movie Oppenheimer.

Fig. 2. The similarity relationships of four user requests differ in both semantic space and intent space. In the semantic space, User Requests 1 and 2 both reference the movie "Ready Player One", while the content of User Requests 3 and 4 is associated with "Oppenheimer". However, in the intent space, User Requests 1 and 4 both pertain to the "video-app" intent domain, whereas User Requests 2 and 3 express the intent to utilize the "search-engine".

module performs semantic and intent encoding on it, and then retrieves historical user request data with semantic and intent similarities from the two vector databases, along with annotated task planning examples, which are placed into the prompt. Additionally, based on the intent classification results, we dynamically assemble the task description for the LLM in the prompt. This approach not only significantly reduces the length of the context input for LLM, thereby making more efficient use of limited context length and improving decoding efficiency, but also guides LLM to focus on task planning within the potential intent space, reducing the error rate in task planning for a large intent space.

Overall, our contributions can be summarized as follows:

1. We introduce a straightforward yet effective OSAgent that, through the design of a memory module comprising semantic vector repository and intent vector databases, exhibits enhanced proficiency in managing complex tasks, encompassing hundreds of APIs from thirty types of tasks across the four domains: AI capabilities, third-party applications, system settings, and data resources.

2. By dynamically generating task descriptions in the prompt based on predicted intent domain results, our approach not only significantly reduces the length of the LLM input context but also makes more efficient use of the limited context length, enhancing decoding efficiency. Moreover, it provides focused guidance to the LLM in task planning within the potential intent space, thereby reducing the error rate associated with large-scale task planning.

3. We conducted comparative evaluations of OSAgent's effectiveness using seven distinct LLM models. Additionally, we conducted a comprehensive analysis of the impact of individual modules on performance. This analysis provides valuable insights for the design of models aimed at assisting humans in the execution of complex tasks, based on LLM-based Agents.

II. Related Work

The work of improving natural language understanding with large-scale pre-training began with BERT [20]. BERT employs a encoder structure of the transformer [21] with a billion-scale parameters and is pre-trained on more than 3 billion words, significantly enhancing language understanding capabilities and advancing the state of the art for eleven natural language processing (NLP) tasks. The GPT series [2], [22], [23]is a decoder structure of the transformer. Although it is not as strong as BERT in understanding capability at the same scale, it exhibits strong language generation ability after brute force parameter enhancement, as well as in-context learning ability. Since the learning loss of GPT is to predict the next word, it is inconsistent with the human-expected response generation. To solve this mismatch problem between pre-training LLM and human-expected response, OpenAI proposed the InstructGPT

[3]. After fine-tuning with manually annotated instruction data, the model demonstrates strong human-computer dialogue capabilities, which are the basis for ChatGPT[1].

Benefiting from Meta's open-sourced large language model LLaMA [7], both industry and academia have quickly followed up on the research and application of LLMs. For example, Taori et al. [24] fine-tuned LLaMA and proposed Alpaca just two weeks after the release of LLaMA, improving the performance from 68% to 76% compared to ChatGPT. Furthermore, Chiang et al. [25] fine-tuned the LLaMA model and presented Vicuna within two weeks, boosting performance to 92%. Although open-source models like LLaMA have significantly reduced the entry barrier for massive AI models, the latest assessment results from the Vicuna team [26] indicate that the gap in capabilities between open-source large models and GPT in areas like Math, Coding, and Reasoning remains substantial.

As a medium of information exchange, the massive amount of text data obtained from the Internet is inherently open-domain and multi-task-oriented. Therefore, it is hoped that language models pre-trained on extensive text data could potentially pave the way for the realization of artificial general intelligence (AGI). At the current stage, there are two possible research trends leading to AGI. One is to attempt to build large-scale AI models capable of handling multi-modal and multi-task situations, while the other is to adopt a hybrid architecture that combines centralized LLMs with specialized smaller models.

For the first research trend, DeepMind constructed a generalist agent [8] that attempts to address more generalized multi-modal perception and cognitive decision-making problems through the sequence prediction capabilities of language models. This encompassed 604 tasks, including dialogue generation, image description, game decision-making, mechanical control, and more, aiming to achieve a model's capability to handle multiple modalities and domains. However, it incurred high training costs and faced challenges in achieving comprehensive performance breakthroughs compared to specialized models.

More work is leaning towards the second research trend, which is an Agent framework with a LLM as the central controller and various specialized small models in different subdomains as sub-functions. The LLM is responsible for connecting with users through natural language interaction, and then decomposing and planning tasks while invoking various sub-functional modules. Examples of this paradigm include TaskMatrix.AI [26], HuggingGPT [14], and AutoGPT [15]. This hybrid architecture combining a central control model with specialized small models leverages the strengths of large-parameter language models and domain-specific expert models, offering strong controllability. However, these Agents still face performance challenges when dealing with complex tasks. For instance, HuggingGPT [1] can manipulate AI models but only supports a limited number of huggingface model types and three input types, including text, image and audio, with an accuracy rate of only 46%. AutoGPT's primary contribution is its concept, but its performance is unstable, prone to entering infinite loops, and supports a limited range of executable task types.

Based on the aforementioned direction, this paper is devoted to improving the existing LLM-based Agent framework and designing an OSAgent model that can tackle complex tasks with high accuracy.

[1]https://chat.openai.com/

III. Our OSAgent

The main purpose of OSAgent is to assist users in more conveniently controlling the operating system through natural language. It employs a large language model as its core for understanding user intents and task planning based on user requests. OSAgent can also invoke various types of tools within the operating system, including AI capabilities, third-party applications, system settings, and data resources. Its framework primarily consists of three parts: the LLM-based agent, external memory, and OS tools, as illustrated in Figure 3. When OSAgent receives a user request, it generates a prompt based on vector search results and intent classification results. This prompt is subsequently handed over to the LLM for the purpose of task list planning, taking into consideration potential dependencies among these tasks. LLM manipulates OS tools using a JSON-formatted language, and the external memory significantly enhances LLM's in-context learning ability, enabling it to handle complex tasks effectively. Finally, some results are directly responded to the user by the OS tools, while others are organized by LLM and generated as responses to the user.

A. Tool API and Task Description

In order to facilitate the seamless invocation of OS tools by OSAgent, we have undertaken the design of a unified and standardized API interface for hundreds of APIs from thirty types of exemplary tasks across the four domain of tools, as illustrated in Table I. It is evident that the parameters associated with AI capabilities are relatively straightforward, primarily involving text, audio, and image data. Conversely, the parameters and action spaces for the other three types of tools are more diverse and extensive, posing a greater challenge in task planning for the LLM-based Agent.

Based on a standardized API interface, LLM operates on OS tools using a JSON-formatted language structured as follows: [{"task" task, "id" task_id, "dep" dep_id, "args" {"arg" arg_value or <GENERATED>-dep_id}}]. In this JSON structure, "task" represents the task name, which is the task derived from understanding and analyzing user requests. "id" denotes the task number, starting from 0. In the case of multiple tasks being split, the "id" for each task would be assigned as 0, 1, 2, and so on. "dep_id" must be within the "dep" list. The "dep" field signifies the task dependencies, where the dependent tasks provide input data for the current task. In cases where there are no dependencies, the "dep" value is set to [-1]. The special tag "<GENERATED>-dep_id" refers to the content produced by dependent tasks. Different tasks have

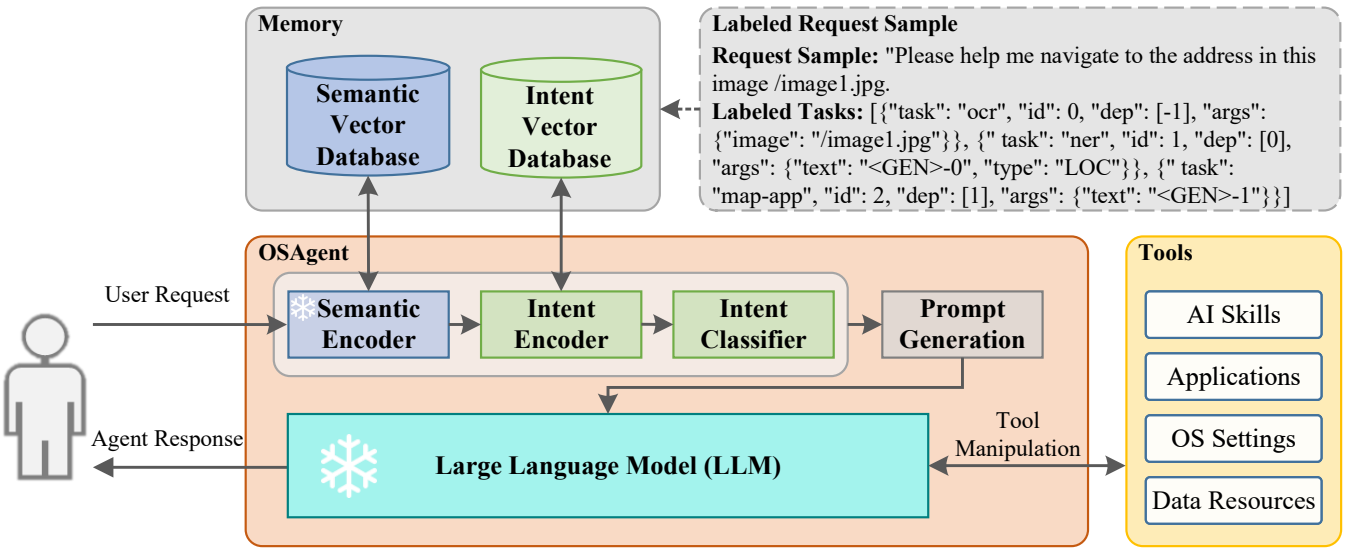


Fig. 3. The framework of our OSAgent. LLM serves as the core controller to manipulate operation system tools by planing a list of tasks based on a list of tasks based on the user request. The external memory stores historical user requests, along with corresponding semantic vectors, intent vectors, and human-annotated task planning instructions.

varying "args" parameters, as shown in Table I. Consequently, describing all types of tasks in detail would result in lengthy prompts, potentially exceeding the context length limit of the LLM.

B. Encoder and Memory

LLMs have strong in-context learning capabilities, meaning they can learn how to complete tasks from demonstrations in the prompt without updating LLM's parameters. However, the quality of the demonstration significantly affects the effectiveness of LLM in-context learning. To address this issue, we set up an external memory module to store the semantic vectors and intent vectors of historical user requests separately, as well as their corresponding labeled task planning instructions, i.e., JSON-formatted language, as shown in Figure 1. The two vector databases are constructed based on a BERT+LSTM architecture that projects user requests into semantic and intent spaces, as illustrated in Figure 4.

In the encoder framework, the BERT structure adopts the standard BERT-base structure with 110M parameters. It is initialized with the parameters from the text2vec-base [27] and frozen. During the training process, only the parameters of the LSTM are updated. The two encoded vector spaces can further enhance the matching of user requests with similar intents. When a new user request is encoded by the Encoder, more relevant data can be retrieved from the vector library to improve the quality of the demonstration.

C. Intent Classifier and Training Strategy

After obtaining the intent vector, we subsequently employed a linear neural network-based intent domain classifier to project the vector dimensions onto intent domain category dimensions (i.e., 30 dimensions, consistent with the number of tasks, as referenced in Table I). The purpose of designing this classifier is twofold: firstly, to update the parameters of the intent encoder through the cross-entropy training loss of the intent domain classification, and secondly, to facilitate content reduction based on the task labels obtained from the intent domain classification as a basis for prompt task descriptions.

To augment the training dataset, we randomly interleaved manually annotated single-task samples with multi-task samples. Additionally, we designed a class-balanced data sampler to mitigate issues related to imbalanced data scales across multiple categories.

D. Prompt Generation

Following HuggingGPT [14], the complete prompt format is as follows: "{task_description} {demonstration} {user_request}". If all tasks related to OS tools are described and demonstrated, the length of task_description will be quite extensive, as depicted in Table II. Using the GPT-4 tokenizer (cl100k_base), the token count for task_description alone reaches 1,697. As our experiments were conducted on a Chinese mobile operating system, the prompts used are in Chinese. To facilitate understanding, we offer an English translation version in Table II, as well as in the following sections. Similarly, when carefully selecting 20 examples for the demonstration to comprehensively cover all tasks, the token count for the demonstration also reaches 2,020. Finally, when the original prompt is combined with the user_request, the average token length would reach 3,754.2 using the GPT-4 tokenizer.

To refine the prompt, we initially replace the carefully curated set of 20 examples in the original prompt with semantically and intent-wise similar user request samples retrieved from memory. While this approach does not reduce

TABLE I

DESCRIPTION OF THE UNIFIED AND STANDARDIZED API INTERFACE, INCLUDING TASK NAMES, PARAMETER DESCRIPTIONS, AND CONFIGURABLE PARAMETER VALUES. OCR IS THE ABBREVIATION OF OPTICAL-CHARACTER-RECOGNITION.

–	Task Name	Arguments
AI Capabilities	speech-recognition	audio
	entity-recognition	text, type: [“PERSON-NAME”, “ID”, “TEL”, “CAR”, “LOC”, “MAIL”, “URL”]
	image-segmentation	image, type: [“human”, “cat”, “car”, “dog”], action: [“extract”, “delete”]
	OCR	image
	speech-synthesis	text, type: [“boy”, “girl”]
	image-generation	text
	summarization	text
	translation	text, source_lang: [“zh-CN”, “EN”], target_lang: [“zh-CN”, “EN”]
	image-caption	image
Applications	app-manage	action: [“open”, “close”, “install”, “uninstall”] , app: [app name]
	map-app	content, action: [“search”, “navigation”], app: [app name]
	e-store	content, action: [“search”, “check-order”, “purchase”], app: [app name]
	chat-app	content, action: [“search”, “send-message”, “voice-call”, “video-call”], app: [app name]
	music-app	content, action: [“search”, “play”], app: [app name]
	search-engine	content, app: [app name]
	video-app	content, action: [“search”, “play”], app: [app name]
	social-media-app	content, action: [“search”, “post”, “follow”], app: [app name]
	food-review-app	content, action: [“search”, “book”], app: [app name]
	food-delivery-app	content, action: [“search”, “order”], app: [app name]
OS Settings	setting-switch	action: [“open”, “close”], setting: [“brightness”, “volume”, “ringtone”], value
	degree-adjust-setting	action: [“up”, “down”, “adjust”], setting: [about 20 settings, e.g., “silent-mode”, “talkback”]
	alarm	action: [“add”, “close”, “countdown”], time: [SQL-like expression]
	calculator	formula: [SQL calculation expression]
	voice-call	action: [“search”, “call”, “hang-up”, “answer”], contact
Data Resources	data-note	keyword, scope: [“title”, “content”], action: [“add”, “search”, “semantic-search”, “delete”, “semantic-delete”], date: [SQL-like expression]
	data-calendar	keyword, scope: [“title”, “content”], action: [“add”, “search”, “semantic-search”, “delete”, “semantic-delete”], date: [SQL-like expression]
	data-sms	keyword, scope: [“title”, “content”], action: [“add”, “search”, “semantic-search”, “delete”, “semantic-delete”], date: [SQL-like expression], contact, type: [“inbox”, “outbox”, “draft”]
	data-email	keyword, scope: [“title”, “content”], action: [“add”, “search”, “semantic-search”, “delete”, “semantic-delete”], date: [SQL-like expression]
	data-document	keyword, scope: [“title”, “content”], action: [“add”, “search”, “semantic-search”, “delete”, “semantic-delete”], date: [SQL-like expression]
	data-photo	keyword, action: [“add”, “search”, “semantic-search”, “delete”, “semantic-delete”], date: [SQL-like expression]

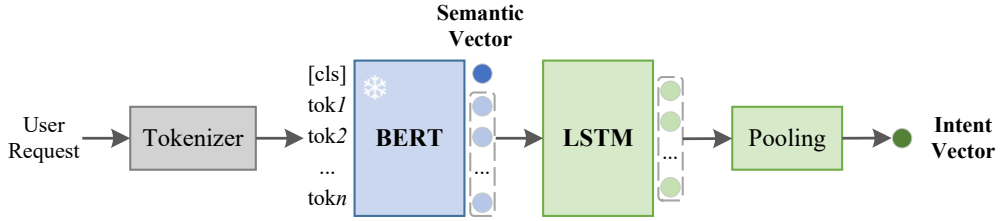


Fig. 4. Framework of joint semantic and intent encoder. Given a user request as input, it first goes through the Tokenizer for tokenization, resulting in n tokens (denoted as tok1˜tok$n$). A [cls] token is padded in front and then fed into BERT for encoding. The output encoding vector corresponding to [cls] serves as the semantic vector, and the remaining token vectors are sequentially input into the LSTM model. The output of the LSTM is subjected to pooling to obtain the intent vector. Each entity circle in the figure represents a vector.

the token length, it significantly enhances the effectiveness of the demonstration section, better guiding the relevance of the in-context learning data for LLM. Additionally, in our experiments, we also compare the method of constructing demonstrations by matching similar data examples through memory to assess its potential to reduce reliance on the number of examples.

On the other hand, to streamline the task_description, we dynamically generate task descriptions based on the intent domain classification results, incorporating only the relevant tasks into the prompt. This approach significantly reduces the token length of task_description. For instance, if the intent classification results are "data-note", "summarization", and "chat-app", the length of task_description would be reduced from $1,696$ to 628 tokens.

E. Task Planing with LLM

In our experiments, the parameters of the LLM were frozen. Therefore, we are able to directly invoke any open-source or commercial LLM API for task planning. The prompt generated in the previous chapter served as the input to the LLM, and the generated JSON-formatted language served as the basis for task planning.

Differing from HuggingGPT [14], our approach standardizes task API specifications, relieving the need for considering model selection for each task. However, it remains necessary to parse resource dependencies among multiple tasks. Within our JSON format, we employ the "dep" argument and the special tag "<GENERATED>-dep_id" to reference content generated by dependent tasks, as outlined in the JSON format description provided in Table II. When multiple tasks exhibit dependency relationships, task execution is delayed until the output results from preceding tasks are available as inputs for subsequent tasks. Conversely, in the absence of inter-task dependencies, multiple tasks can be processed in parallel.

IV. EXPERIMENTS

A. Setup

In our experiments, we utilized seven state-of-the-art commercial Language Model (LLM) platforms, including GPT-4, GPT-3.5-turbo[2], GeminiPro[3], MiniMaxPro[4], WenXinYiYan[5], XunFeiXiongHuoV2[6], and SenseTime8K[7]. Despite being excellent commercial LLMs, they are all accessible to the public through their respective APIs. All the experimental results below were obtained using the API version released in December 2023. Due to the requirement of a temperature greater than 0 for some LLM APIs, we uniformly set the decoding temperature to 0.1 across all of them to ensure more stable outputs from the LLMs.

[2]GPT-4 and GPT-3.5: https://platform.openai.com/
[3]GeminiPro: https://deepmind.google/technologies/gemini/#introduction
[4]MiniMaxPro: https://api.minimax.chat/
[5]WenXinYiYan: https://yiyan.baidu.com/welcome
[6]XunFeiXiongHuoV2: https://xinghuo.xfyun.cn/
[7]SenseTime8K: https://chat.sensetime.com

The semantic encoding module employs a standard BERT-base structure, consisting of 12 transformer blocks with a hidden size of 768, resulting in a parameter count of 110 million. The intent encoding model utilizes a unidirectional LSTM with a hidden feature dimension set at 512. The intent domain classification model is implemented as a linear neural network with an output dimensionality of 30. The parameters of the semantic encoding model are initialized and fixed using text2vec, with the training process exclusively updating the model parameters of intent encoder and intent domain classifier. The model is trained with a learning rate of 0.01 and a batch size of 128.

We manually annotated task planning instruction data for $2,109$ user requests, with $2,009$ of them used as the training set, comprising $1,723$ single-task requests and 286 multi-task requests. The average number of tasks in multi-task samples is 2.21. As mentioned in Section 3.3, we randomly shuffled the training set samples to augment the size of multi-task samples and selected 10% for the evaluation set. The test set consists of 100 samples, including 50 single-task requests and 50 multi-task requests.

B. Result

1) Performance Comparison with the Baseline: Compared to the original prompt style used in HuggingGPT [14], our approach based on external memory matching shows a significant improvement in task planning capability. As shown in Table III, taking GPT-4 as an example, the ACC using the Original Prompt (HuggingGPT style) is only 56%, but the token size of the prompt reaches 3,754.2. In order to reduce the token size of the prompt, we attempted to use prompts that do not include the task description and prompts that do not include examples separately, but both methods resulted in a significant performance degradation. Among these, the prompt without demonstration performed worse, with GPT-4 achieving an ACC drop from 56% to 30% and an edit distance increase from 9.2 to 43.6. After semantic vector matching and intent vector matching, there was a noticeable improvement in performance, with GPT-4 achieving an ACC increase to 80% and an edit distance reduction to 3.4. Refining the task description through intent domain classification significantly reduced the token size of the prompt, with minimal degradation in ACC and edit distance performance. This consistent performance trend across most LLMs demonstrates the significant enhancement in task planning capabilities by OSAgent for controlling various types of system tools.

2) Performance Comparison of Various LLMs: In Table III, we can observe variations in the performance of different LLM variants. Notably, GPT-4 exhibits a significantly superior capability for task planning across various aspects when compared to other LLMs. For example, when utilizing the Original Prompt (w/o demonstration), a robust language comprehension ability is essential for precise task planning based solely on task descriptions in response to user requests. In this scenario, GPT-4 outperforms other LLMs by an absolute margin of 8% in terms of ACC score, while ACC scores for other LLMs

TABLE II
A FULL TASK DESCRIPTION IN THE ORIGINAL PROMPT, INCLUDING THE INPUT ARGUMENTS AND POSSIBLE VALUES.

As the natural language understanding module of the AI assistant, you are required to decompose user requests into multiple tasks. Please provide a single-line JSON output: [{"task" task, "id" task_id, "dep" dep_id, "args" {"arg" arg_value or <GENERATED>-dep_id}}]. "task" represents the task name, which is the result of understanding and planning user requests into various tasks. There are four types of tasks: AI capabilities (including ["speech-recognition", "entity-recognition", "image-segmentation", "optical-character-recognition", "speech-synthesis", "image-generation", "summarization", "translation", "image-caption"]), third-party applications (including ["app-manage", "map-app", "e-store", "chat-app", "music-app", "search-engine", "video-app", "social-media-app", "food-review-app", "food-delivery-app"]), system settings (including ["setting-switch", "degree-adjust-setting", "alarm", "calculator", "voice-call"]), and data resources (including ["data-note", "data-calendar", "data-sms", "data-email", "data-document", "data-photo"]). The "id" represents the task sequence number, starting from 0. If multiple tasks are split, the "id" numbers for these tasks are 0, 1, 2, and so on. "dep_id" must be in the "dep" list. The "dep" field indicates the tasks on which the current task depends, and these dependent tasks provide input data to the current task. If there are no dependencies, the "dep" value is [-1]. The special tag "<GENERATED>-dep_id" refers to the content generated by dependent tasks.

Task "speech-recognition" has the "args" parameter containing ["audio"] that can depend on a url or other task output. Tasks "entity-recognition", "speech-synthesis", "image-generation", "summarization", and "translation" have "args" containing ["text"] that can also depend on a url or other task output. "entity-recognition" additionally includes the "type" parameter with values ["PERSON-NAME", "ID", "TEL", "CAR", "LOC", "MAIL", "URL"]. "speech-synthesis" also includes the "type" parameter with values ["boy", "girl"]. "translation" includes the "source_lang" and "target_lang" parameters with values ["zh-CN", "EN"].

Tasks "image-segmentation", "optical-character-recognition", and "image-caption" have the "arg" parameter containing ["image"] that can depend on a URL or other task output. "image-segmentation" additionally includes the "type" and "action" parameters with values ["human", "cat", "car", "dog"] and ["extract", "delete"] respectively.

Task "app-manage" has the "args" parameter containing ["action", "app"], where "action" can have values ["open", "close", "install", "uninstall"].

Task "search-engine" has the "args" parameter containing ["content", "app"], where "content" can depend on a URL or other task output.

Tasks "map-app", "e-store", "chat-app", "music-app", "video-app", "social-media-app", "food-review-app", and "food-delivery-app" have the "args" parameter containing ["content", "action", "app"], where "content" can depend on a URL or other task output. "map-app" has "action" values of ["search", "navigation"]. "e-store" has "action" values of ["search", "check-order", "purchase"]. "music-app" and "video-app" have "action" values of ["search", "play"]. "social-media-app" has "action" values of ["search", "post", "follow"]. "food-review-app" has "action" values of ["search", "book"]. "food-delivery-app" has "action" values of ["search", "order"]. "chat-app" has "action" values of ["search", "send-message", "voice-call", "video-call"] and includes an additional parameter "contact."

Task "setting-switch" has the "args" parameter containing ["action", "setting"], where "action" can have values ["open", "close"], and "setting" can have values ["silent-mode", "do-not-disturb-mode", and so on].

Task "degree-adjust-setting" has the "args" parameter containing ["action", "setting", "value"], where "action" can have values ["up", "down", "adjust"], and "value" can have values ["max", "min", percentage such as "20%"].

Task "alarm" has the "args" parameter containing ["action", "time"], where "action" can have values ["add", "close", "countdown"], and "time" is represented as precise time using "hh:mm" for hours and minutes or relative time using "NOW() + HOUR(h) + MINUTE(m) + SECOND(s)".

Task "calculator" has the "args" parameter containing ["formula"], which is expressed as an SQL formula value.

Task "voice-call" has the "args" parameter containing ["action", "contact"], where "action" can have values ["search", "call", "hang-up", "answer"], and "contact" can depend on other task outputs.

Tasks "data-note", "data-calendar", "data-sms", "data-email", and "data-document" have the "args" parameter containing ["keyword", "scope", "action", "date"], where "keyword" can depend on a URL or other task output, "scope" can have values ["title", "content"], "action" can have values ["add", "search", "semantic-search", "delete", "semantic-delete"]. "data-sms" and "data-email" additionally include the "contact" and "type" parameters, with "type" values of ["inbox", "outbox", "draft"].

Task "data-photo" has the "args" parameter containing ["keyword", "action", "date", "location"], where "action" can have values ["search", "call", "hang-up", "answer"], and "keyword" can depend on other task outputs.

All data resources related tasks include the "date" parameter, represented as precise dates in YYYY-MM-DD format or relative dates using CURDATE() +/- offset, with operations like YEAR, MONTH, WEEK indicating years, months, and weeks. For example, WEEK(CURDATE()) - 1 represents the previous week. Multiple constraint units are connected with AND, and if it is a range, it is separated by a comma. "-INF" represents the earliest historical date, and "INF" represents the furthest future date.

Dependent URLs can be in the form of /text1.txt, /audio1.wav, /image1.jpg, and other file paths. If it is not possible to obtain the values of the parameters from the user request, they can be directly assigned as "None." If the "type" parameter has multiple values, they can be separated by semicolons. For example, the "entity-recognition" task for personal information can have multiple type combinations: "PERSON-NAME;ID;TEL;MAIL".

TABLE III

PERFORMANCE COMPARISON BETWEEN OSAGENT AND THE BASELINE. ACC REPRESENTS ACCURACY, WHERE HIGHER VALUES INDICATE BETTER PERFORMANCE. ED DENOTES EDIT DISTANCE (USING LEVENSHTEIN DISTANCE HERE), WHERE LOWER VALUES INDICATE BETTER PERFORMANCE. THE PROVIDED PROMPT EXAMPLES CONSIST OF A CAREFULLY SELECTED SET OF 20 EXAMPLES, ENCOMPASSING ALL TYPES OF TASKS. HOWEVER, DUE TO THE CONTEXT LENGTH LIMITATIONS OF THE LLM, THESE EXAMPLES MAY NOT COVER ALL PARAMETER TYPES FOR EVERY TASK. TOKEN SIZE IS THE AVERAGE PROMPT LENGTH CALCULATED USING THE GPT-4 TOKENIZER AND DOES NOT REPRESENT THE ACTUAL TOKEN SIZE FED TO EACH LLM. OP AND RP REPRESENT ORIGINAL PROMP AND REFINED PROMPT RESPECTIVELY. OP-1: ORIGINAL PROMPT (HUGGINGGPT STYLE), OP-2: ORIGINAL PROMPT (W/O TASK DESCRIPTION), OP-3: ORIGINAL PROMPT (W/O DEMONSTRATION), RP-1: REFINED PROMPT (W/ SV), RP-2: REFINED PROMPT (W/ IV), RP-3: REFINED PROMPT (W/ SV+IV), RP-4: REFINED PROMPT (W/ SV+IV+IC). SV: SEMANTIC VECTOR, IV: INTENT VECTOR, IC: INTENT DOMAIN CLASSIFIER. W/: WITH, W/O: WITHOUT.

–	Tokens	GPT-4		GPT-3.5		GeminiPro		MiniMax		WenXinYiYan		XunFeiXingHuo		SenseTime8K	
		ACC↑	ED↓	ACC↑	ED↓	ACC↑	ED↓	ACC↑	ED↓	ACC↑	ED↓	ACC↑	ED↓	ACC↑	ED↓
OP-1	3754.2	56%	9.2	26%	64.1	52%	19.0	47%	21.7	N/A*	N/A*	11%	71.3	16%	53.2
OP-2	2057.2	39%	13.4	25%	56.2	31%	26.5	33%	26.2	18%	46.31	N/A**	N/A**	10%	122.7
OP-3	1734.2	30%	43.6	10%	147.5	22%	62.7	10%	64.6	2%	108.7	1%	180.4	0%	203.0
RP-1	3700.2	80%	3.4	59%	20.3	68%	8.3	71%	7.6	N/A*	N/A*	51%	35.1	34%**	56.9**
RP-2	3480.1	81%	2.8	57%	20.6	66%	10.3	69%	6.1	N/A*	N/A*	38%	24.9	34%**	51.8**
RP-3	3676.1	83%	2.7	55%	27.1	71%	7.5	69%	4.8	N/A*	N/A*	46%	19.3	33%**	43.5**
RP-4	2425.5	83%	3.6	54%	30.4	71%	4.4	72%	5.6	37%*	96.3*	52%**	24.9**	29%**	45.1**

* Partial or all samples exceed the maximum context length limit of the LLM API.
** Partial or all samples' misidentifying of sensitive content resulting in response errors.

remain below 10% (except for GeminiPro). Furthermore, in most cases, GPT-4 outperforms GeminiPro in ACC and edit distance metrics. It should be noted that the context length of WenXinYiYan is relatively short, leading to token size exceeding length constraints in the Original Prompt (HuggingGPT style) scenario. Both XunFeiXingHuo and SenseTime8K exhibit issues with the misidentifying of sensitive content, resulting in incorrect responses in certain test scenarios.

3) The Performance Impact of the Number of Examples: Considering that OSAgent generated demonstration through external memory matching possess stronger exemplar qualities, we aim to investigate here whether it is feasible to reduce the originally designed example dataset to further decrease the token size of prompts. Table IV presents the performance of four LLMs, namely GPT-4, GPT-3.5, GeminiPro and MiniMax, under varying numbers of examples. It can be observed that, apart from the marginal performance improvement achieved by increasing the number of examples in GPT-4, additional examples do not result in significant performance gains for other LLMs. In fact, having a smaller number of examples appears to provide some benefits. For instance, in the case of GPT-3.5, when provided with 12 examples, the peak performance is an ACC of 59%. In contrast, MiniMax achieves its highest performance, an ACC of 74%, with just 8 examples. This may be attributed to the fact that a small subset of samples with the highest semantic and intent similarity already provides effective exemplars. However, the use of only 2 examples in extreme cases does not yield the optimal results. On the other hand, a reduction in the number of examples further compresses the token size of the prompt. For instance, the average token size of prompts with 8 examples provided decreased from the original 2,425.5 to 1,306.9. Since the length of context can influence the decoding speed of LLMs, reducing the token size of prompts is crucial in human-machine interaction scenarios where response latency is of paramount concern.

V. DISCUSSION

The proposed OSAgent, based on the LLM, demonstrates strong language comprehension and task planning capabilities on various tasks in the mobile operating system. OSAgent exhibits an enhanced ability to comprehend complex user intents described in natural language. Nevertheless, despite this progress, our experimental results, when compared to seven state-of-the-art commercial Language Model (LLM) platforms, indicate areas for improvement. The primary directions for enhancement include: 1) updating the parameters of the LLM during the model learning process to enhance the OSAgent's task planning capability for user requests based on foundational language comprehension abilities; 2) considering the possibility of deploying the LLM at the edge for OS task planning, significantly reducing user control latency and network signal dependency; 3) correcting task planning errors or ambiguities of the OSAgent through multi-round human-machine interactions.

VI. CONCLUSION

In this article, we propose an OSAgent utilizing a JSON-formatted language to manage hundreds of APIs from thirty types of tasks across the four domain of tools, encompassing AI capabilities, third-party applications, system settings, and data resources. We fully leverage the in-context learning ability of LLM by designing an external memory module that enables the model to recall semantically and intention-wise similar examples from a annotated database to serve as demonstrations for prompts. Additionally, we employ intent domain classification results for each user request to dynamically generate task descriptions within prompts, reducing the token size. Experimental results demonstrate a significant enhancement in

TABLE IV

PERFORMANCE ACROSS VARYING NUMBERS OF EXAMPLES. IN THIS TABLE, 50% OF THE EXAMPLES ARE RETRIEVED USING SEMANTIC VECTORS, AND 50% USING INTENT VECTORS, WHICH ARE THEN RANDOMLY MIXED AND INSERTED INTO THE DEMONSTRATION. IN REFERENCE TO TABLE III, ORIGINAL PROMPT IN THIS TABLE REPRESENTS THE ORIGINAL PROMPT (HUGGINGGPT STYLE) IN TABLE III, WHILE REFINED PROMPT REPRESENTS THE REFINED PROMPT (W/ SV+IV+IC).

–	Tokens	GPT-4		GPT-3.5		GeminiPro		MiniMax	
		ACC↑	ED↓	ACC↑	ED↓	ACC↑	ED↓	ACC↑	ED↓
Original Prompt	3754.2	56%	9.2	26%	64.1	52%	19.0	47%	21.7
Refined Prompt (2 examples)	695.8	70%	6.1	40%	42.0	56%	10.7	65%	10.9
Refined Prompt (8 examples)	1306.9	81%	2.6	50%	29.4	68%	5.8	74%	5.7
Refined Prompt (12 examples)	1693.7	82%	2.6	59%	24.1	75%	4.8	71%	6.1
Refined Prompt (20 examples)	2425.5	83%	3.6	54%	30.4	71%	4.4	72%	5.6
Refined Prompt (30 examples)	3307.4	85%	2.0	56%	21.4	73%	4.9	69%	6.4

the Agent's task planning accuracy (ACC) and a reduction in edit distance by leveraging demonstrations retrieved from the external memory. The dynamic generation of task descriptions within prompts substantially reduces token size without a pronounced decline in LLM's task planning capability in some LLMs like GPT-4 and MiniMax. Notably, the task planning performance even improves after compressing the prompt's token size, for instance, in models such as WenXinYiYan and XunFeiXingHuo. The OSAgent showcases the potential of LLM-based agents in controlling operating systems, thereby paving a new pathway towards enhancing human-computer interaction systems.

REFERENCES

[1] R. OpenAI, "Gpt-4 technical report," *arXiv*, pp. 2303–08 774, 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," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[3] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 730–27 744, 2022.

[4] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[5] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[6] A. Zeng, X. Liu, Z. Du, Z. Wang, H. Lai, M. Ding, Z. Yang, Y. Xu, W. Zheng, X. Xia *et al.*, "Glm-130b: An open bilingual pre-trained model," *arXiv preprint arXiv:2210.02414*, 2022.

[7] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[8] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg *et al.*, "A generalist agent," *arXiv preprint arXiv:2205.06175*, 2022.

[9] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin *et al.*, "A survey on large language model based autonomous agents," *arXiv preprint arXiv:2308.11432*, 2023.

[10] J. Andreas, "Language models as agent models," *arXiv preprint arXiv:2212.01681*, 2022.

[11] G. V. Aher, R. I. Arriaga, and A. T. Kalai, "Using large language models to simulate multiple humans and replicate human subject studies," in *International Conference on Machine Learning*. PMLR, 2023, pp. 337–371.

[12] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[13] C.-M. Chan, W. Chen, Y. Su, J. Yu, W. Xue, S. Zhang, J. Fu, and Z. Liu, "Chateval: Towards better llm-based evaluators through multi-agent debate," *arXiv preprint arXiv:2308.07201*, 2023.

[14] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang, "Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface," *arXiv preprint arXiv:2303.17580*, 2023.

[15] H. Yang, S. Yue, and Y. He, "Auto-gpt for online decision making: Benchmarks and additional opinions," *arXiv preprint arXiv:2306.02224*, 2023.

[16] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.

[17] A. Bulatov, Y. Kuratov, and M. S. Burtsev, "Scaling transformer to 1m tokens and beyond with rmt," *arXiv preprint arXiv:2304.11062*, 2023.

[18] J. Ding, S. Ma, L. Dong, X. Zhang, S. Huang, W. Wang, and F. Wei, "Longnet: Scaling transformers to 1,000,000,000 tokens," *arXiv preprint arXiv:2307.02486*, 2023.

[19] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," *arXiv preprint arXiv:2307.03172*, 2023.

[20] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of NAACL-HLT*, 2019, pp. 4171–4186.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[22] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[23] 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.

[24] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, "Alpaca: A strong, replicable instruction-following model," *Stanford Center for Research on Foundation Models. https://crfm. stanford. edu/2023/03/13/alpaca. html*, vol. 3, no. 6, p. 7, 2023.

[25] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, I. Stoica, and E. P. Xing, "Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality," March 2023. [Online]. Available: https://lmsys.org/blog/2023-03-30-vicuna/

[26] L. Zheng, W.-L. Chiang, Y. Sheng, and H. Zhang, "Chatbot arena leaderboard week 8: Introducing mt-bench and vicuna-33b," June 2023. [Online]. Available: https://lmsys.org/blog/2023-06-22-leaderboard/

[27] X. Ming, "text2vec: A tool for text to vector," 2022. [Online]. Available: https://github.com/shibing624/text2vec