LLMind: Orchestrating AI and IoT with LLM for Complex Task Execution

Hongwei Cui, Yuyang Du, Qun Yang, Yulin Shao, Soung Chang Liew

Abstract

Task-oriented communications are an important element in future intelligent IoT systems. Existing IoT systems, however, are limited in their capacity to handle complex tasks, particularly in their interactions with humans to accomplish these tasks. In this article, we present LLMind, a large language model-based (LLM-based), task-oriented AI agent framework that enables effective collaboration among IoT devices, with humans communicating high-level verbal instructions, to perform complex tasks. Inspired by the functional specialization theory of the brain, our framework integrates an LLM with domain-specific AI modules, enhancing its capabilities. Complex tasks, which may involve collaborations of multiple domain-specific AI modules and IoT devices, are executed through a control script generated by the LLM using a Language-Code transformation approach, which first converts language descriptions to an intermediate finite-state machine (FSM) before final precise transformation to code. Furthermore, the framework incorporates a novel experience accumulation mechanism to enhance response speed and effectiveness, allowing the framework to evolve and become progressively sophisticated through continuing user and machine interactions.

Introduction

Task-oriented communications and execution framework are an important trend to exploit artificial intelligence to allow IoT systems to interact with humans to execute complex tasks [1, 2]. They enable human users to control multiple IoT devices simultaneously through various media, including text, voice, video, and virtual reality. Communications between humans and IoT devices, and among IoT devices, are indispensable in such systems. Traditional communications systems emphasize on throughput, latency, and reliability. For intelligent IoT systems, beyond these requirements, a shift is needed to also emphasize "intention" at the high level [3]. Hence, the entrance of Large Language Model (LLM).

The article proposes and demonstrates the efficacy of a framework, LLMind, that incorporates LLM to allow humans to interact with IoT devices, and for IoT devices to communicate and collaborate with each other, to perform complex tasks.

LLMind not only reaffirms LLM's impressive linguistic proficiency and exceptional logical reasoning [4], but also showcases its ability to transform conventional IoT devices into an overall IoT system that can accomplish complex tasks through collaborations with high-level instructions from humans. In particular, in LLMind, LLM serves as an orchestrator to facilitate seamless intention-oriented communications among human and IoT entities to execute complex tasks.

The LLM orchestrator in LLMind goes beyond rigid scripted intelligence in traditional IoT device control methods. It endows IoT devices with advanced intelligence, enabling seamless coordination among diverse devices to achieve complex tasks that were previously unattainable using conventional approaches. By harnessing the in-context learning capability of the LLM, the agent can achieve swift adaptation and optimal resource utilization, leading to efficient task solutions. Consequently, users can effortlessly express their needs without concerns about operational details.

Integrating LLM into IoT device control presents a number of challenges that need to be tackled before its potentials can be realized. First, currently, a general LLM cannot effectively handle specialized tasks like object detection or facial recognition, which can already be well performed by traditional AI models in specific domains. Thus, a challenge lies in combining the power of specialized but fragmented AI modules with LLM to handle various complex tasks. Furthermore, due to LLM's slow inference speed and high computational costs, it is crucial to investigate methods that can enhance the system's response speed and overall efficiency. Another significant challenge involves the efficient control of IoT devices through task-oriented communications by the LLM-based AI agent. The LLM must accurately learn the diverse functionalities and operational characteristics of various IoT devices. Additionally, the design of efficient interaction mechanisms that enable users to naturally and intuitively express their needs to the LLM-based AI agent presents a further challenge. Our framework effectively overcomes the aforementioned challenges.

The contributions of our work are manifold. First, inspired by the functional specialization theory of the brain, which states that different regions

Hongwei Cui, Yuyang Du, Qun Yang, and Soung Chang Liew (corresponding author) are with The Chinese University of Hong Kong, Hong Kong SAR; Yulin Shao is with University of Macau, Macau S.A.R. Hongwei Cui, Yuyang Du, and Qun Yang contributed equally to this work.

of the brain are dedicated to specific cognitive functions or tasks, enabling efficient and optimized processing of information, a key feature of our framework is its use of LLM to integrate domain-specific AI modules, forming an overall intelligent system. This synthesis enables the general-purpose LLM to effectively execute specific subtasks by invoking specialized AI modules to accomplish an overall complex task, enhancing system flexibility and practicality. Moreover, by incorporating new AI modules, the AI agent acquires new capabilities and continues to evolve to become ever more capable.

Second, to enhance the response speed and overall efficiency of the agent, we employ a novel method to classify and locally store control scripts (LLM-generated plans) for specific user commands. By utilizing an NLP AI model to identify similar instructions in the records, even when they have varying tones or expressions, the agent can retrieve validated solutions, thus avoiding the need for complete replanning. This approach allows the system to accumulate experience and progressively improve its proficiency through interactions.

Third, the agent leverages the LLM for complex task planning and the generation of the corresponding control script using a novel finite-state machine approach, enabling efficient and precise invocation of AI modules and IoT devices. This method significantly improves accuracy and success rates.

Last but not least, for ease of use, we connect our demo system to WeChat, a widely used social media platform, to allow ordinary non-tech savvy users to conveniently interact with the LLM agent.

We refer to the intelligence level attained by our framework as Artificial Complex Intelligence (ACI), which serves as a bridge between Artificial Narrow Intelligence (ANI) [5] and Artificial General Intelligence (AGI) [6], as illustrated in Fig. 1. ACI integrates the capabilities of various ANI modules and undergoes continuous evolution toward AGI. In summary, our framework goes beyond mere device control and provides a comprehensive blueprint for a future task-oriented, intelligent, and collaborative IoT device ecosystem empowered by LLM.

FRAMEWORK DESIGN

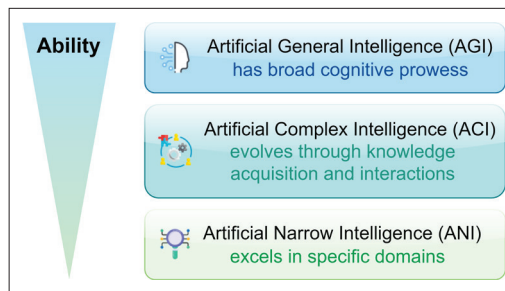Our framework, depicted in Fig. 2, consists of five components: user interface, LLM, coordinator, AI

FIGURE 1. Artificial Complex Intelligence (ACI).

modules, and IoT devices. The operational workflow of a demo system is illustrated in the flowchart presented in Fig. 3.

Users can interact with the system by sending text messages through social media software. The LLM promptly acknowledges the messages. For casual chats and greetings, the system provides text replies without further processing. However, for specific requests and commands, the system formulates plans and strives to fulfill the user's instructions using integrated AI modules and connected IoT devices.

After receiving a user instruction, the system strives for efficiency by first searching for validated and feasible historical scripts stored locally in the coordinator's experience archive. This process utilizes NLP techniques to locate previously issued user commands with similar semantic meanings. If a matching control script is found, it is executed accordingly. However, in the absence of a matching control script, the LLM undertakes the planning process and generates a new control script to invoke AI modules and IoT devices.

The system performs preliminary feasibility validation on the newly generated script, checking for syntax errors or hardware-related issues. If the script fails to execute correctly, the system updates the context (the coordinator's context repository), gathers error information, and triggers the LLM to regenerate the script based on the updated context. If the script continues to fail after multiple retries (more than three attempts), the system collects error information and reports it to the user. Conversely, if the script executes successfully, upon completion, the system summarizes the intermediate results gathered during script execution and generates an appropriate response based on the user's instructions.
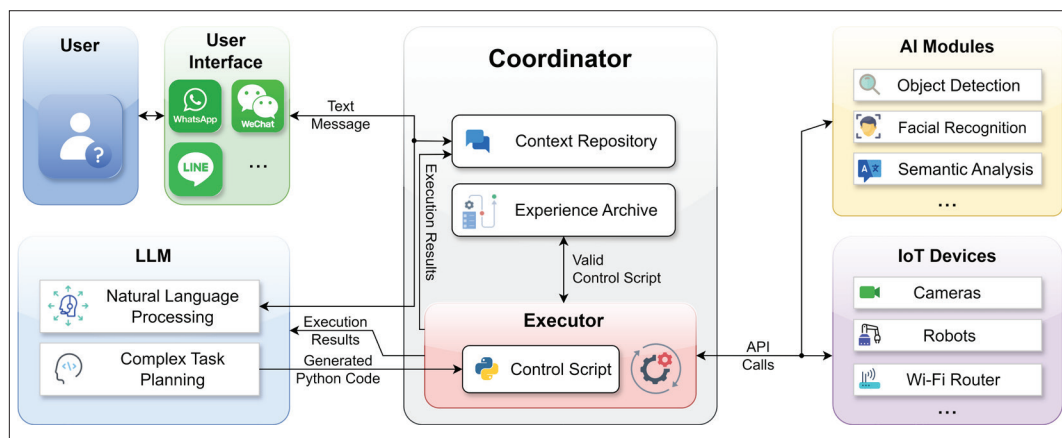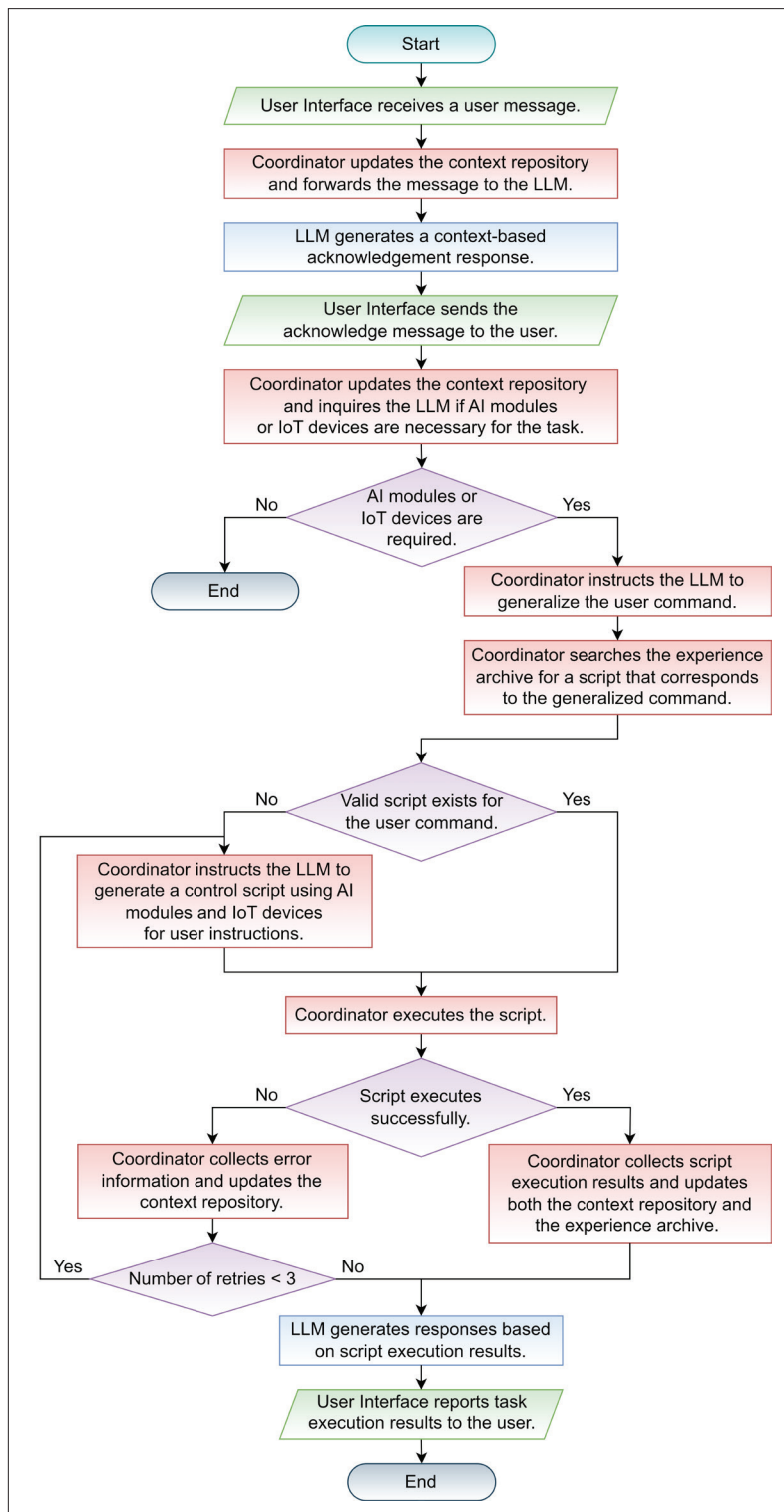

FIGURE 2. System diagram.

FIGURE 3. System operational process flowchart.

LLM

The LLM in our framework performs two key functionalities. First, it engages in conversations with users, generating contextually appropriate responses. To enhance the lifelike and immersive response of the LLM, we adopt a role-playing technique based on the LLM's in-context learning ability. In our demo system, the LLM assumes the persona of a housekeeper, seamlessly blending into our everyday scenarios.

Second, the LLM does the planning and generates control scripts for user tasks utilizing specialized AI modules and IoT devices as part of the available resources and methods. Notably, the LLM itself does not directly interact with AI modules and IoT devices. Instead, it generates control scripts to invoke AI modules and send control commands to IoT devices through network connections. This design enables the system to adapt to different AI modules and IoT devices without requiring extensive modifications to the LLM itself, enhancing scalability and interoperability. We use Python as the programming language for our demo system due to its wide adoption in the field of artificial intelligence.

LLM Role-Playing: The LLM role-playing approach is supported by carefully crafted system-level prompts, as shown in Algorithm 1.

It is important to note that in order to optimize the user experience, distinct user profiles are necessary for tailoring responses to individual users. These user-related details enable the system to provide personalized interactions, catering precisely to each user's preferences and needs.

FSM-Based Language-Code transformation using LLM: Within this framework, we propose a Language-Code transformation scheme using a finite-state machine (FSM) as a strategic and justified step for generating scripts. The complexity of user tasks often necessitates breaking them down into manageable subtasks. The FSM acts as an intermediate step, providing a structured approach to connect and sequence these subtasks. This choice is inspired by the inherent ability of FSMs to represent complex processes through a set of states and transitions, ensuring a systematic and organized generation of control scripts. Our Language-Code transformation scheme tackles two critical issues.

First, a user may describe the task he wants the AI agent to do with simple (sometimes ambiguous) human language, but the successful execution of a task relies on complex code descriptions that precisely define the operation of AI modules and IoT devices. The mismatch between high-level human command and precise code description necessitates a reliable Language-Code transformation that can be refined iteratively until the human command is realized to produce outcomes satisfactory to the human.

To address the first issue, the scheme expands the information in the human command by leveraging knowledge about the environment. For instance, when a user instructs the agent to fetch a bottle of iced Coke, the agent should understand that iced Cokes are stored in the refrigerator, and the refrigerator is located in the kitchen corner, even if not explicitly mentioned. In our system, we expand the command's information with pre-stored environment knowledge using in-context learning.

Second, the LLM must precisely transform the expanded information (expressed in natural language) into executable code. However, the language description can become lengthy after expansion to enhance clarity, and as a result, the generated code may become excessively complex, posing a challenge to error-free execution. Directly using the LLM to generate complex code from long language descriptions is unreliable due to its limited ability to handle intricate code with advanced functionality, as observed in [7]. This may result in

the loss of crucial information from the human command or render the generated code unusable.

To address this problem, we use FSM to bridge the gap between language and code descriptions. First, we utilize the task decomposition abilities of the LLM to break down the overall task into sub-tasks, representing them as states in the FSM. The interconnection between these sub-tasks is captured through transitions between FSM states. Next, we utilize the LLM to convert each sub-task within the FSM states into executable Python code. Finally, by combining the code descriptions of each state and incorporating state-transition conditions, we obtain the final code for the overall task.

Compared to the "Language-Code" transformation, our novel "Language-FSM-Code" transformation scheme maintains precision and conciseness in conveying information since an FSM can precisely and comprehensively describe a task [8]. Additionally, by providing APIs that enable the LLM to reference and interact with AI modules and IoT devices, the complexity of sub-task transformation within each FSM state is manageable for current LLMs. Notably, our scheme enhances reliability without causing information loss.

Figure 4 shows the Language-Code transformation procedure, involving three phases: context expansion, Language-FSM transformation, and FSM-Code transformation.

For context expansion, the system expands the task description using the environment information stored in the context repository of the coordinator. This enables a precise and executable task description.

The Language-FSM transformation uses the expanded task description and API information to generate the FSM representation of the task. Providing API information helps the LLM understand the capabilities of AI modules and IoT devices, leading to a reasonable and efficient FSM design. Without API information, the FSM representation may include complex sub-tasks or redundant processes, causing inefficiencies.

The FSM-Code transformation is divided into three sub-phases. An FSM has two components: state operation and state transition. We generate code descriptions for each component in separate sub-phases, merging them in the final sub-phase. Our method simplifies the LLM's task and improves reliability compared to a direct Language-Code transformation.

In the first sub-phase, we provide the LLM with the FSM derived in the Language-FSM transformation and API information. This enables the LLM to generate state operation code that incorporates the desired functionality. In the second sub-phase, example codes are provided to assist the LLM in understanding Python-based state transitions. Finally, the LLM is provided with the FSM, code generated in the previous sub-phases, and supplementary reference codes to facilitate the production of the final code.

Coordinator

The coordinator in our framework acts as a central hub, connecting and enabling the seamless operation of system components. Without a central coordinator, the system components would operate in isolation, leading to inefficiencies and potential conflicts. The coordinator ensures
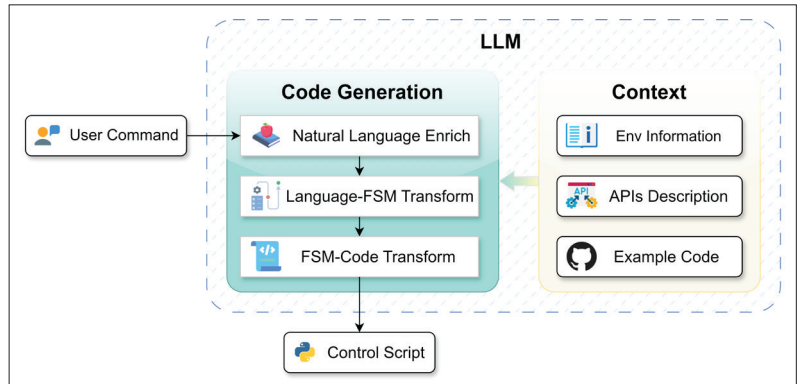
ALGORITHM 1.

FIGURE 4. Language-Code transformation procedure.

smooth functioning of the agent system. It comprises three major parts: context repository, experience archive, and script executor.

The context repository contains the contextual information provided to the LLM for both conversation and code-generation purposes. This information serves as a crucial resource for the AI system, providing the necessary knowledge and understanding of user interactions, system states, and task requirements. It comprises four main components:

• Environment Information: A list of available AI modules and IoT devices with their function descriptions, as well as the installation locations of IoT devices. This enables the LLM to be aware of the available AI modules, IoT devices, and their capabilities.

• API Descriptions: Comprehensive descriptions of the API functions used to invoke AI modules and control IoT devices within the script, encompassing details such as inputs, outputs, specifications, expected time costs, and other relevant information. This enables the LLM to effectively leverage the available functionality.

• Chat History: Records of previous conversations between the system and the user, allowing the LLM to refer back to previous interactions.

• Execution Results and Error Reports: Execution results of generated scripts and error reports returned by the script executor. This information helps the LLM assess the outcome of script execution and identify issues or errors that may have occurred.

The context is updated in real-time based on user interactions and the execution status of control scripts, ensuring that it is readily available for intelligent decision-making and adaptive behavior within the AI agent system.

Num.	Conversations	Contents
[1]	User → Housekeeper	Can you help me to count the number of people in the room and identify who they are?
[2]	Housekeeper → User	I can help you count the number of people in the room and identify them. Please give me a moment to gather the information.
[3]	Assistant → Housekeeper	Do you require any assistance?
[4]	Housekeeper → Assistan	Yes, please. I need assistance with counting the number of people in the room and identifying them.
[5]	Assistant → Housekeeper	Please repeat the employer's instruction using as few words as possible.
[6]	Housekeeper → Assistant	Count people in room, identify them.
In the absence of a valid script in the experience archive, the LLM generates a script, which is then executed in the executor.		
[7]	Assistant → Housekeeper	We have found 5 people in the room. The list of identified 3 people are ['Mike', 'Ada', 'Joe']. There are also 2 people that can not be identified. Their locations are shown in the following list in a format of [x, y]: [[10, 1], [12, 5]].
[8]	Housekeeper → User	There are 5 people in the room. Three of them have been identified as Mike, Ada, and Joe. The other 2 people cannot be identified and their locations are at [10, 1] and [12, 5].

TABLE 1. Dialog contents between different roles.

The experience archive serves as a database for previously generated scripts. Valid scripts are stored locally and linked to their corresponding user instructions. This enables the system to reuse preexisting scripts when a similar instruction is given by the user in the future, resulting in improved efficiency and faster response speed. Importantly, the agent's ability continues to grow through the accumulation and generalization of experience. However, a challenge lies in how the system evaluates the similarities between user instructions. Our system implements a semantic analysis NLP model that maps words to a vector space [9]. The semantic similarity is then determined by computing the distance between these vectors. To enhance the search speed, we vectorize the instruction associated with the script before storing it in the database.

The control script generated by the LLM is passed to the executor, which provides the necessary runtime environment and offers API functions that can be called within the scripts. Furthermore, the executor monitors hardware status, handles runtime data, generates execution reports, and updates both the context repository and the experience archive.

AI MODULES

Currently, LLM cannot effectively handle many domain-specific tasks. To address this limitation, we integrated domain-specific AI modules into the framework, aiming to enhance overall performance by assigning specialized tasks to dedicated modules, resulting in improved capabilities, accuracy, and performance beyond LLM's general knowledge.

The individual AI modules can be well versed in various specialized AI techniques, such as computer vision, natural language processing, speech recognition, sentiment analysis, recommendation systems, and more. Each module is trained and optimized for its specific task, enabling it to provide highly accurate and specialized results.

Furthermore, AI modules enable the AI system to adapt and evolve rapidly in response to emerging challenges and advancements in specific domains. New modules can be developed and integrated into the system as needed, expanding its capabilities and keeping pace with the evolving landscape of AI technologies.

Overall, the aggregate of the LLM and specialized AI modules, orchestrated by a coordinator, forms a general-purpose AI system capable of performing complex tasks. In a way, the system mimics the fact that the biological brain is also divided into coordinated parts with specialized functions.

IoT DEVICES

To ensure efficient control of diverse IoT devices in our framework, manufacturers must provide well-documented API functions. These functions should encapsulate the necessary functionality, enabling the AI agent to initiate specific actions or retrieve information from the IoT devices.

It is important to emphasize that manufacturers should prioritize security considerations when exposing API functions for remote control, thereby preventing unauthorized access or tampering.

USER INTERFACE

Our framework is open to various interaction methods, relying on valid LLM input or methods to convert user input into a suitable format for LLM processing. Social media software is a good interface, seamlessly aligning with our daily lives and providing a natural interaction experience.

EXPERIMENTS

We have verified the feasibility of the framework through experiments. In the experiments, the user sends a text message to the AI agent. It then formulates a solution by examining the user's command and the system's available resources and methods, including AI modules and connected IoT devices. Following that, the AI agent generates an executable script and responds to the user's request based on the execution outcome.

We deployed the coordinator and AI modules of the system on an edge server. The edge server is connected to the same local network as the IoT devices via WiFi and can interact with ChatGPT using OpenAI's APIs.

The available IoT devices in the system are as follows:
• Security cameras: The cameras are positioned within the room to capture images of the surrounding environment. These images are transmitted to the edge server via

a local network. Our experiment employs three cameras: two placed at the corners of the room, approximately 2 meters above the ground, and the third mounted on top of a TurtleBot mobile robot.

- TurtleBot mobile robot: The TurtleBot mobile robot can receive target position coordinates through a network connection and autonomously plan paths to navigate within the room, intelligently avoiding obstacles.
- WiFi router: The WiFi router can manage bandwidth allocation for each device and enforce restrictions on their maximum network speed.

Additionally, the system integrates the following AI modules:

- Object detection: We employ the YOLO v8 model [10] to detect various objects in the scene, including people.
- Face recognition: We use an open-source project called Face Recognition [11] to identify individuals in the scene.

SCENARIO 1: CHECK-IN AND SECURITY

Guests arrive at a party. The user sends a message to the AI agent: "Can you help me count the number of people in the room and identify who they are?" (Demo video for scenario 1: https://youtu.be/3AI5qRntEEU) Table 1 provides the dialogue content between different roles in the experiment. Among them, the LLM acts as the housekeeper, while the coordinator serves as the assistant.

In step 2, the LLM, acting as the housekeeper, responds to the message sent by the user.

In step 3, the coordinator questions the LLM about the need for assistance. Following each round of prompt and response interaction between the user and the LLM (similar to steps 1 and 2), the coordinator repeats the same question to the LLM. In the scenario where the LLM assumes the role of a remote housekeeper who relies on assistants in the user's residence to carry out tasks. This inquiry serves to verify if the user has given specific instructions. Ambiguous instructions, as well as unrelated casual conversations, will cause the system to prompt the user to provide additional information or be filtered out. When the LLM indicates a need for help, it signifies that the user has assigned a task that could potentially be handled using AI modules and IoT devices.

To ensure effective communication between the LLM and the coordinator, the LLM's response in step 4 must incorporate specific keywords that facilitate word matching. These rules are already embedded in the context and are not shown in Table 1. Specifically, when the LLM's response is affirmative and assistance is needed, it should include the keyword "Yes." Conversely, if the response is negative, it must include the keyword "No." In this experiment, the LLM's response is affirmative, based on user's instruction.

In steps 5 and 6, the coordinator prompts the LLM to generate a summary of the user's instructions to facilitate the search for relevant historical scripts stored in the experience archive. This search process involves comparing the semantic content of the user's instructions. However, the original task description may be poorly specified and be mixed with various linguistic habits. Additionally, the LLM may require multiple interactions with the user to fully grasp the details of specific
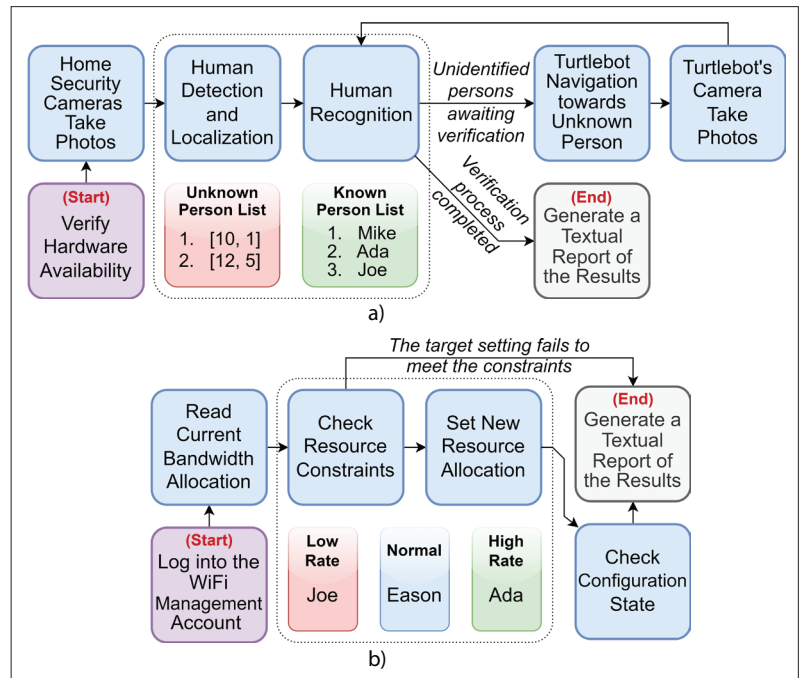

FIGURE 5. FSM of generated code: a) Check-in and security scenario; b) Network management scenario.

tasks. To handle these conditions, in step 5, we request the LLM to provide a concise summary of the user's instructions. Subsequently, we conduct a search for relevant scripts based on the LLM's summary to enhance accuracy.

In this experiment, the coordinator could not find a suitable control script in the experience archive for the task. Consequently, the LLM is instructed to generate a new script. The code is generated using an FSM approach, as depicted in Fig. 5a. In addition to relying on the security camera installed on the ceiling, if the person cannot be identified due to low photo resolution or incomplete facial information, the AI agent will then instruct the TurtleBot to move to the position of the unknown person to take a close-shot photo for further assurance. The code generates two lists that store the execution results, one for known persons and another for unknown persons.

In step 7, once the script execution is complete, the LLM receives the execution results of the API functions. Throughout the script execution process, API functions may generate log information, encompassing intermediate results, warnings, and errors. This log information is collected and sent to the LLM as context for analysis.

Finally, in step 8, the LLM generates a report based on the user's question and the script's log information and sends it to the user.

SCENARIO 2: NETWORK MANAGEMENT

Our framework is designed to handle both one-off complex tasks and scenarios that involve multiple interactions with users. Next, we consider a scenario where a family of three individuals shares the same home WiFi network (Demo video for scenario 2: https://youtu.be/aTGD8EjQ8kM). The interaction process between the AI agent and the user in this scenario is similar to the one described in the check-in and security scenario.

In this scenario, Eason, the user, is watching a movie, and asks the AI agent, "Can you improve my

internet speed? My movie has a slight lag." When the AI agent, encounters this task for the first time, it generates an FSM based on the available APIs of the WiFi router, as depicted in Fig. 5b. Following the transition rules defined in this FSM, the LLM generates executable scripts to address the request.

It is important to note that in this experimental scenario, we limit the total bandwidth of the shared network to 100 MB/s for demonstration purposes. Additionally, the speed limit for each user is fixed in a stepped manner, categorized as Low Rate (20 MB/s), Normal (30 MB/s), and High Rate (50 MB/s). In the first attempt, the AI agent successfully uses the network management API to elevate the user's bandwidth from the Low Rate to the Normal level without exceeding the total bandwidth of the shared network. It then informs the user about the adjustment result.

However, if Eason remains dissatisfied with the network speed, he can ask, "Can you increase my internet speed once more?" Since the script for "improving internet speed for Eason" is already stored in the experience archive, the previously generated script is executed directly. However, in this experiment, during the execution of the script, it is determined that further increasing the network speed for Eason is not possible. This limitation arises because another user, Ada, is already in the High Rate list. Increasing the network speed for Eason again would exceed the total bandwidth limit of the shared network. Therefore, this execution is terminated, and the reason is communicated to Eason. The second scenario demonstrates that our framework is capable of handling tasks that involve multiple interactions with users.

Recent Work in the Context of Our Framework

[1] presents a comprehensive task-oriented surveillance framework for virtual emotion informatics to tackle human-requested tasks. This framework places emphasis on the integration of mobile robots, UAVs, and virtual emotion informatics. It highlights the crucial role of task-oriented communication in future wireless networks, a concept that aligns with the focus of our article.

LLM-Planner [12] introduces a few-shot planning method using LLMs for versatile embodied agents that rapidly comprehend and execute natural language instructions. It generates high-level plans adapted to the environment. Our work makes further advances by providing technical implementation details on how to use the LLM for real-world task planning and scheduling, along with a new FSM-based approach that significantly improves task execution success rate.

The framework proposed in [13] combines LLMs, digital twin systems, and industrial automation for intelligent production planning and control. Integrating LLMs with digital twin systems enhances production system scalability and adaptability. Our proposed framework offers an alternative approach by utilizing predefined APIs to control IoT devices, avoiding the complexity and cost of building a digital twin, while still achieving a satisfactory effect.

Conclusion

Our work advances the control of IoT devices by incorporating LLM into a task planning and orchestration framework called LLMind. LLMind seamlessly combines LLMs' language skills, reasoning abilities, and in-context learning with specialized AI modules, expanding the versatility and flexibility of IoT systems. The proposed approaches, including FSM-based control script generation, user interaction, and historical script retrieval mechanisms, contribute to enhancing user experiences and system efficiency, with the system becoming increasingly intelligent as it learns from interactions among LLMs, humans, AI modules, and IoT devices. This comprehensive blueprint for intelligent and collaborative IoT ecosystems sets the stage for future advancements, where LLMs and AI-driven solutions enhance device coordination and convenience in our daily lives harmoniously.

Going forward, we plan to explore the implementation of locally running fine-tuned open-source foundational models, such as Llama, for control script generation. Doing so can potentially improve response speed and bolster privacy and security measures. Additionally, we are intrigued by the idea of adapting multi-modal LLMs to enhance human-machine interaction through voice, video, and virtual reality. We believe that this pursuit not only deserves serious consideration but also holds tremendous potential.

References

[1] S. Lee et al., "Task-Oriented Surveillance Framework for Virtual Emotion Informatics in Polygon Spaces," IEEE Wireless Commun., vol. 30, no. 3, 2023, pp. 104–11.
[2] Y. Shi et al., "Task-Oriented Communications for 6G: Vision, Principles, and Technologies," IEEE Wireless Commun., vol. 30, no. 3, 2023, pp. 78–85.
[3] W. Xu et al., "Guest Editorial: Task-Oriented Communications for Future Wireless Networks," IEEE Wireless Commun., vol. 30, no. 3, 2023, pp. 16–17.
[4] T. Webb, K. J. Holyoak, and H. Lu, "Emergent Analogical Reasoning in Large Language Models," Nature Human Behaviour, vol. 7, no. 9, 2023, pp. 1526–41.
[5] A. Kaplan and M. Haenlein, "Siri, Siri, in My Hand: Who's the Fairest in the Land? On the Interpretations, Illustrations, and Implications of Artificial Intelligence," Business Horizons, vol. 62, no. 1, 2019, pp. 15–25.
[6] B. Goertzel and C. Pennachin, Artificial General Intelligence, vol. 2, Springer, 2007.
[7] M. Asay, "Are Large Language Models Wrong for Coding?" InfoWorld, 05 2023; available: https://www.infoworld.com/article/3697272/are-large-language-models-wrong-for-coding.html.
[8] P. Gladyshev, "Finite State Machine Approach to Digital Event Reconstruction," Digital Investigation, vol. 1, no. 2, June 2004 pp. 130–49.
[9] M. Xu, "Text2vec: Text to Vector Toolkit," https://github.com/shibing624/text2vec, 2023.
[10] G. Jocher, "YOLOv8 Documentation," docs.ultralytics.com, 05 2020; available: https://docs.ultralytics.com/.
[11] ageitgey, "ageitgey/face recognition," GitHub, 06 2019; available: https://github.com/ageitgey/face recognition/.
[12] C. H. Song et al., "LLM-Planner: Few-Shot Grounded Planning for Embodied Agents With Large Language Models," Proc. IEEE/CVF Int'l. Conf. Computer Vision, 2023, pp. 2998–3009.
[13] Y. Xia et al., "Towards Autonomous System: Flexible Modular Production System Enhanced With Large Language Model Agents," arXiv preprint arXiv:2304.14721, 2023.

Biographies

Author biographies were unavailable at presstime.