TaskSense: A Translation-like Approach for Tasking Heterogeneous Sensor Systems with LLMs

Kaiwei Liu[1], Bufang Yang[1], Lilin Xu[1], Yunqi Guo[1], Guoliang Xing[1], Xian Shuai[2], Xiaozhe Ren[2], Xin Jiang[2], Zhenyu Yan[1]
[1]The Chinese University of Hong Kong, Hong Kong SAR, China
[2]Noah's Ark Lab, Huawei Technologies, Hong Kong SAR, China

ABSTRACT

An increasing number of environments, such as smart homes and factories, are being equipped with multiple sensor systems to enable diverse intelligent applications. However, most existing sensor coordination systems require manually predefined rules, limiting their ability to handle flexible and complex tasks. While recent approaches leverage large language models (LLMs) to interact with external APIs, they struggle to fully understand the capabilities and data dependencies of practical sensor systems. This paper introduces TaskSense, a novel system that coordinates multiple sensor systems in response to users' complex queries. TaskSense introduces a sensor language that automatically translates the capabilities and data dependencies of sensor systems into vocabularies and grammar rules that can be understood by LLMs. It then interprets user intentions into executable task plans for sensor systems using this sensor language in combination with LLMs. Meanwhile, TaskSense checks the solvability of user queries and verifies the correctness of task plan dependencies. To further enhance robustness, TaskSense incorporates a dynamic plan execution mechanism that adjusts plans based on real-time feedback from sensor data availability, data quality and execution results. TaskSense is deployed on real-world smart home systems, utilizing six popular LLMs. The system is evaluated across 4 scenarios involving 9 types of sensor systems, over 60 APIs, 170 tasks and 5 types of data modalities. Results show that TaskSense achieves up to 2× higher planning accuracy and a 75% increase in answer accuracy using the similar amount of tokens compared with baseline approaches.

CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile computing**; • **Computing methodologies** → **Planning and scheduling**; • **Computer systems organization** → **Embedded and cyber-physical systems**.

KEYWORDS

Sensor Systems, Large Language Models, LLM Agent, Internet of Things

SenSys '25, May 6–9, 2025, Irvine, CA, USA

ACM ISBN 979-8-4007-1479-5/25/05
https://doi.org/10.1145/3715014.3722070

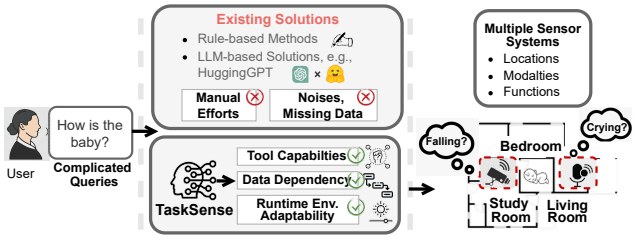

Figure 1: An example scenario of the TaskSense system. It can handle users' complicated queries by coordinating multiple sensor systems.

ACM Reference Format:
Kaiwei Liu, Bufang Yang, Lilin Xu, Yunqi Guo, Guoliang Xing, Xian Shuai, Xiaozhe Ren, Xin Jiang, Zhenyu Yan. 2025. TaskSense: A Translation-like Approach for Tasking Heterogeneous Sensor Systems with LLMs. In *The 23rd ACM Conference on Embedded Networked Sensor Systems (SenSys '25), May 6–9, 2025, Irvine, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3715014.3722070

1 INTRODUCTION

Sensor systems have been widely deployed in various aspects of human life, enabling diverse intelligent applications and tasks such as healthcare and industrial logistics [40, 63]. The market size of sensor systems is forecasted to grow to US$55.2 billion by 2030 [2]. Multiple sensor systems are increasingly deployed and coexist in the same environment to support diverse intelligent applications. Most existing sensor systems are designed for *specialized tasks*, with their functions remaining fixed after deployment [50, 61, 63]. Recently, open-source ecosystems like Home Assistant [9] have become mature, integrating with over a thousand different devices and services. These ecosystems have led to a vision that it is possible to control heterogeneous smart devices from different manufacturers to handle complicated tasks. However, those ecosystems can only support fixed coordination rules, like the automation scripts in HomeAssistant platform, that are manually defined by the human efforts in advance. Thus, these systems are not scalable to handle *flexible and complex tasks* due to too many potential combinations of smart devices.

Recently, Large Language Models (LLMs)-based studies, such as HuggingGPT [49] and ControlLLM [35], have demonstrated that LLMs can understand users' inquires and coordinate multiple AI models for complex tasks. Acting as central controllers, LLMs have the potential to orchestrate heterogeneous sensor systems, overcoming the limitations of traditional AI models restricted to single, predefined tasks. Based on their powerful comprehension capabilities, LLMs can understand flexible and complex queries

expressed in natural language, generate correct plans based on the user's intention, and translate the plan execution results into user-friendly responses. However, existing LLM-based coordination solutions only work with AI models or algorithms, but they struggle to coordinate real sensor systems. *Lacking knowledge of the capability of real-world sensor systems and data dependencies among their functional modules, existing methods fail to accurately judge the solvability of user query and coordinate sensor systems in a correct manner. They also have difficulty maintaining stable performance under the uncertainty factors of real-world environments.* Figure 1 illustrates an example scenario where a home is equipped with a video surveillance system and a smart speaker. The user sends a query to monitor the potential abnormal status of a baby at home. The sensor systems should alert the user if the baby cries when the falling is detected. Such queries require diverse combinations of sensor systems and are highly dynamic. Other examples include, *"Was there anyone in the room yesterday?"* and *"Where are my keys?"*.

To address this research gap, this paper explores the design of an LLM-powered system that coordinates one or more sensor systems by fully understanding their capabilities and the dependencies between the tools involved. The *tool* in this paper refers to a module with a specific functionality within a sensor system, such as data processing and sensor control. However, designing such a tasking system presents several challenges. First, once the sensor systems are deployed, the coordination system must be able to comprehend the diverse functionalities of the tools. These may include specialized AI models, database operations, sensor controls, or data transmission. Understanding the capability limits of each tool, as well as their complex interdependencies, poses a significant challenge for LLMs. For instance, a surveillance camera system might detect unusual behaviors but may not have the ability to identify individuals. Meanwhile, a facial recognition tool might only accept cropped face images as input, rather than full images captured directly by the sensor. Second, changes in real-world environments can significantly affect data quality and availability, potentially causing coordination failures. It is difficult for the coordination system to automatically generate alternative plans by swapping tools dynamically at runtime to complete the same task.

In this paper, we introduce TaskSense, a translation-based approach that enables LLMs to interact with sensor systems using a specialized framework called sensor language. As illustrated in Figure 1, we first develop a novel sensor language to help LLMs comprehend the capabilities and data dependencies of sensor systems. Once these systems are deployed, our approach automatically translates their functionalities into vocabularies and grammar that LLMs can understand. When a user submits a query, TaskSense first assesses its solvability through a solvability check, and then interprets the user's intent based on the sensor language and translates it into an executable plan that sensor systems can carry out. Next, TaskSense performs a grammar check to detect any dependency errors of the plan. During execution, TaskSense employs a dynamic plan adaptation mechanism, adjusting the plan in runtime based on feedback from sensor data quality, data availability, and execution outcomes. This allows the system to maintain high accuracy and adaptability in real-world environments. Finally, TaskSense translates the sensor systems' outputs back into natural language, providing a clear response to the user.

We summarize the contributions of this work as follows:

- We develop *TaskSense*, the first translation-like approach that enables LLMs to coordinate heterogeneous sensor systems to complete complex tasks in user queries.
- We propose a novel *sensor language* that defines *vocabulary sets* and *grammar* that enable LLMs to understand the capability boundaries and interrelationships of functional modules in sensor systems. This helps TaskSense automatically generate executable plans to coordinate sensor systems and ensure task solvability and data dependency.
- We design a dynamic plan adaptation mechanism for sensor system coordination to address environmental uncertainty in runtime. It allows TaskSense to dynamically adjust the plan execution path based on sensor failure, data quality, and execution results.
- We implement TaskSense on 4 datasets, covering 9 sensor systems, over 60 APIs, 170 tasks and 5 types of data modalities. Evaluation results show that TaskSense can achieve up to 3 times the planning accuracy and 1.75 times the responding accuracy using the similar amount of tokens compared with the state-of-the-art solutions.

2 RELATED WORK

2.1 Tasking Sensor Systems

Various smart sensor systems have been integrated into our daily lives [11, 60, 66]. Traditional sensor systems rely on rule-based task execution strategies. ADmarker [39] combines diverse sensors and AI models in elders' homes to detect early Alzheimer's Disease. Kratos+ [51] uses a policy negotiation algorithm to manage multiple sensors, handling various user requests without conflicts. However, these solutions are limited to predefined and fixed tasks and cannot adapt to dynamic user requirements. Recent works utilize LLMs to solve complex tasks in embedded systems. Works in [14, 48] use LLMs to assign tasks to AI modules and IoT devices, meeting diverse user requirements. Sasha [25] employs LLMs to manage home devices and generate action plans in response to open-ended instructions. However, these solutions do not consider environmental uncertainties when tasking sensor systems, limiting their robustness in real-world applications.

2.2 LLM for Task Planning

LLM Agents. Some works propose using LLMs as the decision center to call external tools for solving complex tasks. ToolFormer [45] shows LLMs can learn to use the API of external tools to generate more reliable answers. HuggingGPT [49] and TaskMatrix.AI [33] use LLMs as a controller to propose task solution plans and then select the corresponding tools for each step. To find the best choice from all possible plans, ToolLLM [44] and ControlLLM [35] adopt tree-based and graph-based methods, respectively, for optimal plan searching. However, these systems only focus on calling general tools like the calculator, Wikipedia, and AI models designed for specialized tasks, without considering the complex environments and data dependencies in practical sensor systems.

Improving the Quality of LLM Planning. Self-check [36] uses an additional LLM as a validator to recognize errors in its own

step-by-step reasoning process. ToolLLM [44] and KwaiAgents [42] improve LLM's planning ability by fine-tuning it on a high-quality instruction dataset. In addition, many studies employ feedback-based strategies to improve plan quality. TaskMatrix.AI [33] and Sasha [25] use feedback from users to improve the plan quality of LLMs. Other studies also employ error messages as feedback to enhance plan quality. Voyager [58] and LLMind [14] use program execution errors as feedback to improve the plan quality of LLMs. Some works also attempt to augment LLMs with observations from objective environments [68, 70]. However, these methods either require additional validators for evaluation or rely on execution errors and human feedback to improve the plan quality, without fully utilizing the information from the sensor systems.

2.3 LLMs for Open-ended Question Answering in Sensor Systems

Adapting Sensor Data to LLM Embedding Space. M4 [69] and OneLLM [21] use a large amount of multi-modal sensor data to train adapters. These adapters can map raw sensor data into the embedding space of an LLM, enabling open-ended question answering based on sensor data. However, adapting diverse sensor data modalities into the embedding space of natural language requires a large amount of training data and is difficult to achieve, especially for complex and information-sparse modalities like IMU.

Retrieval and Summarizing. Many works explore LLMs' capabilities in sensor data reasoning within embedded systems [41, 62, 64]. LLMTrack [67] uses Chain-of-Thought [27] prompting to analyze IMU data for trajectory recognition, while LLMSense [41] uses LLMs for high-level reasoning on daily activity logs from human activity recognition models. In addition, some studies utilize retrieval-augmented generation (RAG) techniques [23, 62] to access external knowledge bases such as patients' daily wearable data [65]. However, these systems face two major limitations: some require continuous execution of various task-specific models to update the data log, posing significant challenges in terms of computational resources and flexibility, while others focus primarily on sensor data understanding and reasoning, neglecting a broader range of task types.

3 MOTIVATION

This section examines current methods for tasking sensor systems, involving three steps: generating a tool calling plan based on user queries, executing the plan, and delivering a response from the execution results. Insights from measurement studies on existing approaches and our real-world testbed shape our design goals.

3.1 Planning from Queries

To understand the capability of existing sensor systems in handling real-world user queries, we use a query set to test the state-of-the-art (SOTA) approach, HuggingGPT [49], within a home elderly care scenario. In this scenario, multiple sensor systems, including a home surveillance system, deep action recognition system and sound detection system, are deployed in an elderly person's home to monitor and record their daily activities. We use HuggingGPT to understand clinical queries derived from MDS-UPDRS [20] about human activities and emotions and then generate tool-calling plans.

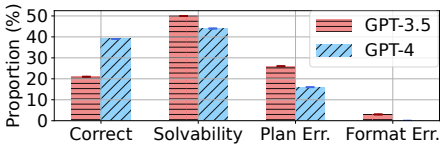

Figure 2: Distribution of planning errors using HuggingGPT for tasks in elderly care scenario.

We construct 38 elderly care queries in total. Given 19 tools from 3 different sensor systems, LLMs need to select from these tools to form a plan for each query.

Figure 2 shows the distribution of errors made by GPT-4 [5] and GPT-3.5 [29], which are mainly from the following aspects:

1. **Misjudging Solvability:** Current approaches often fail to recognize the capability limitation of the toolset and identify queries that exceed its capabilities. User queries that exceed the toolset's capacity usually happen in two cases. First, the toolset may lack a specific function needed for the query. For example, with a query like *"Summarize Bob's blood glucose"*, if the toolset lacks a blood glucose monitoring tool, the system cannot handle it. Second, even if the toolset includes relevant tools, their label setting may not cover the target of the query, such as a HAR tool that includes labels of "Sitting" and "Dancing" but not "Playing Chess". In these cases, instead of generating a misleading plan, the system should notify the user that it cannot fulfill the query. However, accurately understanding the solvability of each query is challenging, as the toolset's capability boundary is complex.
2. **Wrong Plan:** Another major error type involves generated plans that contain incorrect dependencies or tools. Each tool has specific functionalities and distinct input-output requirements, meaning data dependencies between tools are constrained. LLMs may produce infeasible plans by introducing incorrect dependencies. For example, a facial expression recognition tool, which requires face images as input, can only follow a face detection tool in the plan. If it follows a facial emotion recognition or speaker diarization tool, which does not output detected faces, the execution results will be incorrect. Aside from dependency issues, the generated plans may include incorrect tools. These errors occur when LLMs select inappropriate tools from the toolset or generate non-existent tools. For instance, the generated plan for a query about sleep quality might incorrectly include a fall detection tool.

The highest proportion of errors in Figure 2 is due to the models' failure to accurately recognize queries that exceed the toolset's capability boundary. Incorrect dependencies result from the models' difficulty in determining valid dependencies between tools. Besides, wrong tool selection often results from models' challenges in understanding diverse query expression styles. These limitations cause existing methods to struggle to generate high-quality plans.

3.2 Plan Execution and Responding

After developing the execution plan, TaskSense should execute it and respond to the user query. To understand the execution quality of the plan in real-world environments, we conducted experiments in a smart home scenario using the framework of HuggingGPT [49].

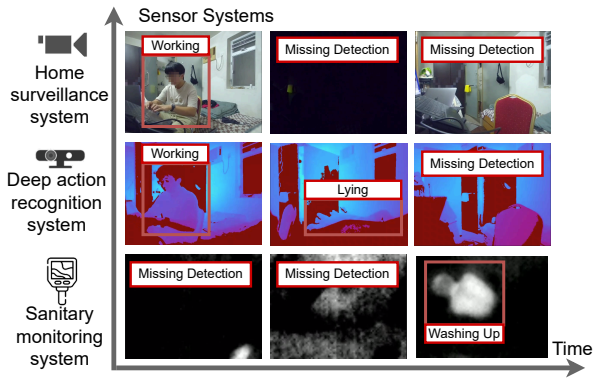

Figure 3: An illustration about environmental impacts on sensor systems. While the LLM planner creates a plan based on user queries, factors such as sensor noise and occlusion can affect the execution results.

We analyzed the execution of plans based on user queries, considering environmental uncertainties that may affect the outcome. In this experiment, we separately execute three different plans on three different sensor systems for the same user query: "*Was Bob overworking today without adequate bathroom breaks?*". Figure 3 demonstrates several periods of the plan execution results. The results show that each plan performs differently when the environment changes. For example, the plan executed on the RGB camera-based home surveillance system fails to produce correct results in the latter two periods due to poor lighting conditions and occlusion. Similarly, the plans executed on the deep action recognition system and sanitary monitoring system do not yield accurate outcomes in the last and first two time periods, respectively, due to occlusion and the subject being out of frame. This demonstrates that although the plan is correct, its execution is highly susceptible to environmental factors. Moreover, the environmental factors vary significantly across different plan execution paths. However, we notice that although some of these systems fail to detect the target in certain time periods, it can still be completed by others, demonstrating a high degree of complementarity among the different sensor systems. Our study highlights three primary types of environmental factors that affect plan execution:

1. **Data Missing:** Tool execution within a plan often requires sensor data input. If a sensor fails due to hardware failure or is accidentally turned off, the required data may not be recorded, leading to failures in the execution.
2. **Data Noise:** Factors like changing light conditions that affect RGB cameras, or noise levels that affect speech capture, can reduce the quality of sensor data. Data noise introduces errors in the plan execution results.
3. **Content-Related Issues:** Issues like objects moving out of the camera frame, occlusions, or individuals in challenging poses cannot be detected by assessing data noise levels alone, but they can also significantly impact the accuracy of results.

These observations indicate the challenges faced by current approaches in handling dynamic environmental factors in real-world settings. To address these issues, leveraging the complementarity between different sensors can be a promising strategy. For instance, while light conditions may affect the performance of the generated plans using RGB camera-based tools, they have minimal impact on the plans using depth camera-based tools. Besides, with multiple human detection sensors placed at various locations and angles in a home, detection results can be guaranteed by switching between plans using different sensors based on the user's location. Therefore, adaptively switching between different plan execution paths can mitigate the impact of environmental uncertainties and enhance the system's overall robustness.

Furthermore, after plan execution, existing approaches also face challenges during the response generation phase. Multiple sensor systems can produce substantial result data in the execution. However, processing extensive data sequences and performing complex computations pose challenges for LLMs to produce responses to the user [72], causing forgetting issues or hallucinations like wrong data association among tools. To understand this issue, we constructed 30 test queries and use GPT-4 to comprehend the plan execution results to generate the answer for each query. Among the test set, there are 5 failed cases due to the incorrect results association among different tools and 3 cases presenting incomprehensive answers. Therefore, it is necessary to implement strategies ensuring the accuracy of the generated answers to address the limitations associated with LLMs.

4 SYSTEM OVERVIEW

In this paper, we introduce sensor language and propose a heterogeneous sensor system tasking approach, TaskSense, which leverages LLMs to coordinate sensor systems for understanding and responding to user queries in dynamic environments. Figure 4 overviews the design of TaskSense. TaskSense first employs an LLM-based sensor system automatic registration approach (§5.1) to initialize the vocabulary set, grammar, and seed examples, reducing the manual effort required for system initialization and updates. When TaskSense receives a query from the user, it first interprets the user's intention based on the sensor language and retrieved examples, translating it into executable tool-calling plans to coordinate sensor systems (§ 5.2). After the coordination plan is generated, TaskSense employs a dynamic plan adaptation mechanism (§5.3) to further adjust the plan execution path in runtime, ensuring they adapt to environmental changes and robust against interferences in embedded systems. Finally, TaskSense translates the sensor language back into natural language feedback based on the plan execution results(§5.4).

5 DESIGN OF TASKSENSE

This section presents the design details of TaskSense. Section 5.1 introduces the necessary components and their initialization process when applying TaskSense into a new scenario, including the sensor language definition and example library. Section 5.2 and Section 5.3 illustrate the plan checking and dynamic plan adaptation techniques, respectively. Section 5.4 introduces the response generation module.

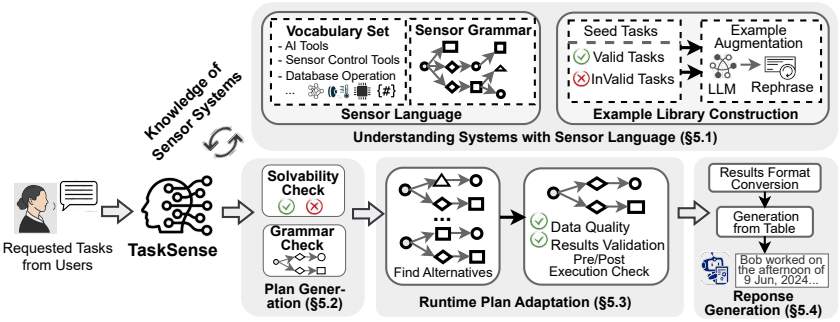

Figure 4: System overview of TaskSense.

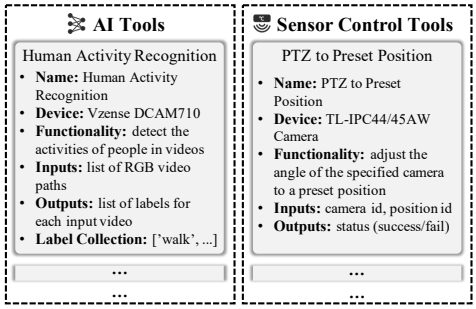

Figure 5: An example of vocabulary set.

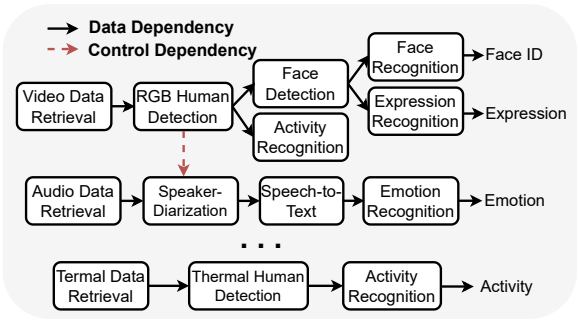

Figure 6: An example of sensor grammar.

5.1 Understanding Systems with Sensor Language

5.1.1 Definition of Sensor Language. In TaskSense, user queries are inputted in the form of natural language, which cannot be directly used to execute the tools in sensor systems. To bridge this gap, we develop a translation-like solution that converts user intentions into executable tool-calling plans using the LLM, which utilizes LLM's impressive ability in language translation tasks [15, 28]. We conceptualize tools and their dependencies as a language, termed *Sensor Language*. In this paradigm, tools are modeled as vocabularies in the Sensor Language, dependencies are modeled as the grammar, and plans are like sentences formed by combining vocabularies following grammatical rules. Sensor Language incorporates two key definitions: vocabulary set and grammar rules. They are included in the LLM prompt with the retrieved examples (Section 5.1.2), enabling the LLM to translate user intents effectively.

Sensor Vocabulary Set. It defines detailed information of all the tools and the basic elements available for the LLM when creating executable plans. As shown in Figure 5, for each tool, each vocabulary includes the following items: category (e.g., small AI model, sensor control), tool name, modalities (e.g., RGB, IMU), devices (e.g., VZense DCAM710), functionality description, inputs (e.g., thermal image), outputs (e.g., label, video), and output label setting (e.g., ["sitting", "running", ...]).

Sensor Grammar. It defines dependency rules among tools that each plan should follow. The sensor grammar consists of a set of directed acyclic graphs (DAGs), where nodes represent tools and edges represent valid dependencies among tools. When included as part of the prompt for LLMs, the graphs will be converted into text format, consisting of node lists (e.g., ["Face Recognition", "Face Detection", ...]) and edge lists (e.g., ["Human Detection -> Fall Derection", ...]). Each valid plan is a subgraph of the sensor grammar. As shown in Figure 6, two types of dependencies exist in the sensor grammar: data dependencies and control dependencies.

Data dependencies define data input-output relationships among tools, where certain tools require outputs from others as inputs, thereby constraining executable plans. These dependencies are used to validate LLM-generated plans (see Section 5.2 for more details). On the other hand, control dependencies, different from data dependencies, mainly define the potential execution preconditions for tools. A control dependency between two tools specifies the conditions under which one tool can execute, which depends on the other one's output. For instance, in Figure 6, the Speaker Diarization tool is triggered only if the RGB Human Detection tool detects at least one person. Once a generated plan passes data dependency verification, control dependencies are automatically inserted into the plan before plan execution.

Sensor vocabulary and grammar are essential for LLMs to translate user queries into executable plans. The vocabulary defines the toolset's capability boundary, while the sensor grammar specifies dependency rules among tools. They will be separately used in the solvability check (§5.2.1) and grammar check (§5.2.2) modules to improve planning quality. When applying TaskSense to a new application scenario, the vocabulary set and grammar need to be constructed first. Existing solutions, like HuggingGPT [49] and Sasha [46], require manual efforts to create a toolset. In contrast, TaskSense achieves automatic initialization. Specifically, TaskSense first constructs the sensor vocabulary set. The user only needs to provide the source codes for software tools (e.g., human detection and activity recognition) and/or a detailed description of the sensor system's APIs provided by the manufacturers (e.g., video capturing and human presence). TaskSense specifies the format of the new vocabulary in the prompt: a list of JSON objects and each JSON object represents a vocabulary. Each JSON object has the following keys: "category", "tool name", "modality", "description", "input", "output" and "label setting". Additionally, TaskSense also provides 5-6 predefined showcases of vocabulary in JSON form to help the LLM understand the format. Then, TaskSense sends the prompt containing the source codes/detailed description of new tools and the showcases to the LLM to generate vocabularies for all the new tools. After constructing new vocabularies, TaskSense starts to construct the new grammar rules. Similarly, it specifies in

the prompt that the grammar rules should be organized as a list of strings. Each string contains the names of two tools connected by an arrow, like "Face Detection -> Face Recognition", representing a dependency. TaskSense also includes a list of strings following this format of grammar rules as the showcase and the vocabularies of the new tools into the prompt. With the prompt, LLMs generate grammar rules for the new tools. After that, TaskSense will then check whether each newly generated grammar rule is valid based on the inputs and outputs of each pair of tools. Users can also add or delete new vocabularies or grammar rules manually.

5.1.2 Example Library. Providing only a vocabulary set and grammar rules is insufficient for learning sensor language. In machine translation, example libraries, known as parallel corpora, are commonly used to organize translation examples for training models [24, 26, 30]. As shown in Table 1, we adapt this approach into an LLM-based method for Sensor Language, designing each example as a query-plan pair. Each query-plan pair includes a user query example and the corresponding plan to solve it. From these examples, LLMs learn to break down user queries into tool-calling steps and organize them as executable plans. The example library serves two key purposes: (i) helping LLMs accurately understand the capability boundaries of the toolset to identify unsolvable queries, and (ii) improving planning robustness to varied styles of user expressions. To achieve the first goal, TaskSense includes both solvable and unsolvable examples in the library. For the second, TaskSense categorizes and samples seed examples, then employs LLMs to expand them automatically.

Seed Example Categorization. We refer to the initially constructed examples as *seed examples*. The primary purpose of seed examples is to ensure broad and diverse coverage in the example library. Given that such diversity is essential for improving instruction tuning performance [12, 71], we propose expanding seed examples through categorical sampling. Possible query categories are organized based on specific applications, and examples are created for each category. For example, categories "activity", "cognition", "emotion" are organized for home elderly care scenario [39]. Within each category, seed examples are organized to be as distinct as possible. Categorization and intra-category diversity enhance the system's performance on unseen user queries.

Solvable and Unsolvable Examples. In real-world scenarios, our system may receive queries beyond the toolset's capability boundary. The system needs to recognize these unsolvable queries, thus preventing misleading responses. However, simply creating examples by category is insufficient for the system to accurately assess query solvability. To address this, we advance the design of the example library by creating both solvable and unsolvable examples for each category. A solvable example includes a processable query and its corresponding plan, while an unsolvable example includes an unprocessable query and an empty plan, indicating that no tool-calling plan is available. This approach significantly enhances LLMs' ability to recognize query solvability.

Example Library Augmentation. Users often exhibit various expression styles, which can affect the accuracy of LLMs' outputs. To address this, we propose using LLMs to diversify the expression styles of queries in the examples. For each seed example, we input the query to an LLM and instruct it to generate several semantically equivalent queries in different styles. Each new query is paired with the seed example's plan to create a new example for the library. Based on the benefit of LLM-based instruction re-writing [32, 59], this approach can effectively augment the example library.

Table 1: Example library in TaskSense.

Type	Query	Planning Result
Activity	Did Bob show repeated action yesterday?	{ "query-video-rgb",..,"human-activity-recognition-rgb", "activities" }
Emotion	How was Bob's mood today?	{ "query-video-rgb",..,"expression-recognition-rgb", "expressions" }
Language	Did Bob mention his travel yesterday?	{ "query-audio-data",...,"speaker -recognition", "speaker ID" }
Unsolvable	Did Bob have a fever or fatigue this week?	{"Unsolvable"}
...	...	...

Similar to the construction of new vocabularies and grammar, TaskSense also automates the construction of seed examples. It specifies that the format of examples should be a list of JSON object pairs. Within each pair of JSON objects, one contains a "role" key with its value as "user" and a "content" key with its value as a user query. The "role" key of another JSON object is "assistant" and the "content" key is the corresponding plan for the user query. This format demonstration is included in the prompt. Like the construction of vocabularies and grammar rules, a few predefined showcases following this format are also included in the prompt by TaskSense. The vocabularies and grammar rules of the new tools generated in the previous steps are included, too. Then, the prompt will be sent to LLMs to generate seed examples for the new tools. This process is performed separately for each seed example class. After generating all new seed examples, TaskSense instructs LLMs to discard invalid or overly similar examples and then performs grammar checking for the rest. It then calculates pairwise semantic similarities among the rest of the examples, and provides ranking based on the similarities as a reference for users to remove redundant examples manually by specifying their IDs.

5.2 Plan Generation

Upon receiving a user query, TaskSense employs a text encoder to generate an embedding of it, which is used to retrieve the most similar examples from the example library based on cosine similarity. TaskSense precomputes and stores embeddings of example queries to reduce the system's latency of each retrieval. To ensure the diversity of retrieved examples, TaskSense retrieves examples evenly across each category. This category-based retrieval strategy enhances diversity among examples, improving the system's generalization by avoiding excessive similarity among the examples. These examples are then combined with the sensor vocabulary, grammar, and user query for plan generation using LLMs. However, plans generated by LLMs may contain errors, such as misjudging the solvability of user queries and incorrect dependencies. TaskSense adopts two key modules to solve these issues: solvability checking and grammar checking.

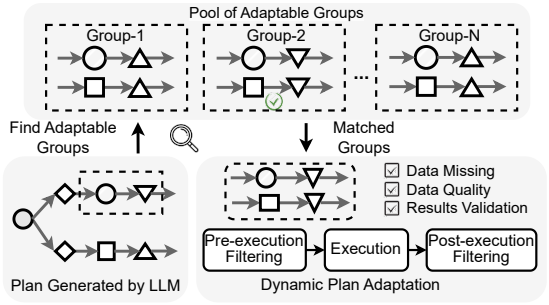

Figure 7: The workflow of dynamic plan adaptation.

5.2.1 Solvability Checking. Some queries are unsolvable because their target labels (e.g., activities, emotions, or events) fall outside the toolset's capability boundary. To handle these unsolvable queries, apart from including unsolvable examples in the example library, we also develop an LLM-based checker to assess queries' solvability based on the toolset's capability limits. Before assessing the solvability, the checker first identifies whether a query is an open-ended query, which refers to those ambiguous queries that do not restrict the response to specific labels, allowing any reasonable response. (e.g., "How do you feel about Alice's exercise routine on Nov 14?"). Open-ended queries will be skipped for further solvability analysis. For queries with specific target labels (e.g., "When did Bob lie on the bed on Nov 13, 2023?"), the checker assesses solvability based on the toolset's capability. It checks if the toolset has the necessary tools for the target functions and, if available, checks whether their label lists include the target labels. Queries targeting labels beyond the toolset's capabilities are marked as unsolvable, and no plans are generated, while the rest pass the check.

5.2.2 Grammar Checking. To identify dependency errors in generated plans, TaskSense uses subgraph matching to verify grammar correctness. TaskSense first converts grammar rules into a directed acyclic graph (DAG) that contains all valid tools and dependencies. The plan is also represented as a DAG, with each node corresponding to a distinct tool. TaskSense then checks if the plan's DAG is a subgraph of the DAG representing the grammar rules. This method not only identifies incorrect dependencies but also flags any mistakenly generated tools not in the predefined toolset.

5.3 Runtime Plan Adaptation

Various dynamics of real-world environments, such as lighting and obstructions, can degrade tool performance. As shown in Section 3.1, static plan execution is vulnerable to such environmental factors. However, the vocabulary set often contains tools with similar functionalities, such as *Facial Expression Recognition* and *Speech Emotion Classification*. Due to this redundancy, multiple feasible plans may exist for the same user query, utilizing different sensor modalities or devices deployed in different locations. Inspired by this, we propose a plan adaptation approach that enables TaskSense to identify replaceable parts of a plan and dynamically select the optimal execution path among alternatives with similar functionalities, based on the runtime quality of sensor data and the outputs of each path. Figure 7 shows the workflow of the dynamic plan adaptation, which is described as follows.

5.3.1 Replaceable Parts Recognition and Alternative Group Matching. First, TaskSense utilizes LLMs to categorize tools in the toolbox by function, grouping similar ones. Then, TaskSense replaces each tool in each group with a plan ending with it by adding the minimal necessary tools it depends on. Such a plan with no redundant tools is called *alternative path*. If multiple alternative paths end with the same tool, they are included in the same group as different paths. All these groups will be gathered together, and we call each group an *adaptable group* and call the group collection *pool of adaptable groups*. At runtime, when a plan is generated, TaskSense traverses the pool of adaptable groups to find groups containing paths that are subgraphs of the generated plan. The matched group is selected as the adaptable group for the corresponding path in the generated plan, as the paths in this group are functionally equivalent to it. If multiple groups match, TaskSense selects the one with the longest path. TaskSense then removes the matched part from the original plan and repeats the process until no more adaptable groups are found. This process converts the LLM-generated plan into a set of adaptable groups and irreplaceable parts.

5.3.2 Pre-execution Filtering. During execution, various real-world factors can affect the availability and quality of sensor data. To select the appropriate paths at runtime, we apply pre-execution filtering before executing each group, considering both data availability and quality. Specifically, for each group with alternative paths, TaskSense first divides the sensor data's time range into equal intervals, e.g., one hour. Then, for each interval, TaskSense examines each alternative path to check if its required sensor data is missing and to assess data quality. If the data is missing or has a significantly low signal-to-noise ratio, that alternative path is skipped for the interval.

5.3.3 Post-execution Selection. Some factors that induce uncertainty cannot be identified by examining missing data or assessing data quality, yet they affect execution results. For example, the occlusion and limited field of view can cause target detection at one location to fail, but other similar tools may work. To address such factors, TaskSense switches to other alternative paths if the current path yields invalid execution results (e.g., empty results). This evaluation serves as feedback to dynamically adjust the plan.

Besides, to reduce the latency of plan execution, TaskSense adopts a cache mechanism. A cache database stores the execution results of each tool on specific data traces. When executing each tool in a plan, the system checks the cache database to see if the output has already been generated.

5.4 Response Generation

Plan execution results are then sent to LLMs for responses generation. As discussed in Section 3.2, there are two main challenges for LLMs in understanding the results: 1. The results exceed the token limits. 2. Generating a response may involve logical understanding and calculation, which LLMs handle poorly. To address these challenges, we propose a formatting method to convert the execution results to a more readable format. TaskSense uses unique identifiers (IDs) to represent the outcome items of each tool and converts the original results to a format that is easily understood by LLMs.

Table 2: Details of datasets.

Datasets	Sensor Modality	# Tools	# Seed Examples	# Queries
In-lab HAR	RGB, Depth	6	8	30
DAHLIA	RGB, Depth	6	8	30
Synthetic	RGB, Depth, Audio, IMU	18	10	50
Real-world	RGB, Depth, Audio, Light, IR, Zigbee, Humidity, Temperature	58	10	60

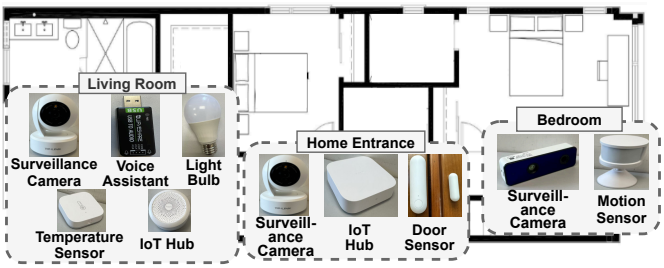

Figure 8: Hardware setup of the real-world dataset.

Outcome-wise ID. To track correspondence among outcome items from different tools, TaskSense assigns a long, randomly generated string as the unique ID to each outcome item of the tools. This ID, along with the tool's original outcomes, is passed to the next tool in the plan as its pre-IDs. As a result, each tool records its input and output arguments, along with the pre-IDs and IDs. By using these IDs and pre-IDs, we can construct result trees.

Results Format Conversion. After obtaining execution results with outcome-wise IDs, we apply the lowest common ancestor matching algorithm [6] to identify objects and their corresponding labels like activity, gender, and identification, constructing the result table. For example, the execution results may contain face and activity recognition according to the sensor grammar DAG shown in Figure 6. By identifying the lowest common ancestor for object detection, we can determine people's activities and emotions. The final formatted results are organized in the form of a table.

Response from Formatted Results. Since the table can exceed LLM token limits and LLMs have limited numerical calculation capabilities [72], TaskSense extract required information from the result tables, similar to SQL queries.

6 IMPLEMENTATION AND DATASETS

6.1 System Implementation

The overall architecture of TaskSense consists of an LLM on the server and multiple tools deployed on the server or edge devices. Sensor data is collected by edge devices. We utilize APIs from Microsoft Azure [37], Amazon Bedrock [8] and Poe [43] platforms to access various LLMs. For the retrieval of examples, we utilize APIs from the Cohere [7] platform for obtaining text embeddings and use the Annoy [52] package to select examples based on similarity.

6.2 Datasets

We evaluate TaskSense on four datasets shown in Table 2:

In-lab HAR Dataset[1]. This dataset for Human Activity Recognition (HAR) is collected in our lab, including data from 30 participants performing 30 different behaviors. Three participants are used for evaluation and others for model training. A Vzense DCAM710 camera system (RGB and Depth) and 6 tools are used in the evaluation.

DAHLIA Dataset [56]. This is an open-source dataset commonly used for HAR. We utilize data from three individuals for evaluation and others for model training. This dataset is collected using three Kinect v2 sensors (RGB and Depth). In the evaluation, we use 6 different tools with the dataset.

Synthetic Dataset. To eliminate the influence of tool accuracy on the performance of TaskSense, we synthesize a virtual dataset using LLMs for mock testing. This dataset contains the behaviors of two individuals over five days by prompting LLMs to generate tool outputs for each individual in every time interval, reasonably simulating their daily activities. These simulated outputs are directly stored in the execution cache. Sensor systems are simulated with failure conditions according to the virtual environment setup. We simulate placing 4 different sensor systems in this virtual environment and use 18 tools in the evaluation.

Real-world Dataset[1]. We collected this dataset in three rooms with 8 heterogeneous sensor systems, including data from 12 individuals. Three individuals' data are used for evaluation. As shown in Figure 8, Room 1 (Living Room) includes a TP-link TL-IPC44AW camera system (RGB), a Waveshare USB to Audio module (Audio), a Xiaomi smart light, and an Aqara temperature sensor. Room 2 (Home Entrance) has a TP-link TL-IPC45AW security camera (RGB) and an Aqara door sensor (Zigbee). Room 3 (Bedroom) includes a Xiaomi motion sensor 2s (IR) and a behavior analysis system using a Vzense DCAM710 camera (RGB and Depth), simulating monitoring for groups with special needs in a bedroom, such as the elderly. 58 tools in total are used for this evaluation.

Since the original datasets lack the corresponding queries, we utilize queries from the real-world surveys and medical questionnaires, including MDS-UPDRS [20], PSQI [10], IPAQ [13], NBI [18], ZBI-C [53], FAS [47], and IADL [31]. Besides, additional queries are created to ensure comprehensive testing. In this way, we construct a specific user query set and toolset for each dataset. For the In-lab HAR and Dahlia datasets, we simulate environmental factors by randomly adjusting videos' brightness and adding Gaussian noise.

Implementation of Tools. We integrate code and documents from various platforms as input for automatic system component construction in datasets. They include TP-link IPC Control APIs [55], Home Assistant [9], Vzense DCAM SDK [57], HuggingFace [17], and FFmpeg [16], as well as public models from GitHub and our own trained models. To enable the execution module to call the tools in a unified form, we encapsulate each tool as a function with standardized input and output formats. Tools in the evaluation mainly include three types: 1. specific AI models (e.g., Fall Detection). 2. Database Operation Tools (e.g., Retrieve Sensor Data). 3. Sensor Control Tools (e.g., Set Light Brightness).

[1]The data collection and study has been approved by the authors' IRB.

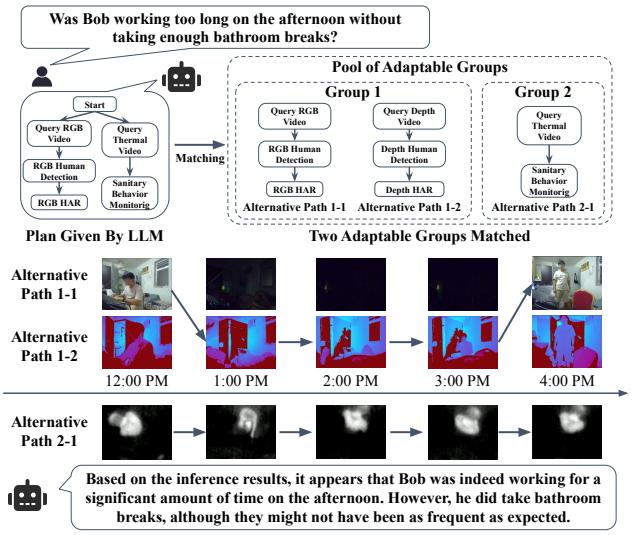

Figure 9: An end-to-end application.

7 EVALUATION

7.1 Evaluation Metrics and Baselines

7.1.1 Evaluation Metrics. We evaluate the overall performance of TaskSense from three dimensions.

Planning Accuracy. This metric measures the portion of correct plans generated by the LLM. The ground truth plan for each query is manually provided, and a generated plan is correct only if its tool selection and dependencies exactly match the ground truth. If multiple valid plans exist for a query, any matching plan is considered correct. Additionally, to enable a more fine-grained comparison between plans, we also define another metric **Planning Score**, which measures the similarity between generated and ground truth plans: $S_{\text{score}} = \frac{1}{n}\sum_{i=1}^{n}\frac{2\,PR_i \cdot RC_i}{PR_i + RC_i}$, where g_i and p_i represent the collections of ending nodes (tool callings without subsequent tool calling) in the ground truth and generated plans, and $RC_i = \frac{|g_i \cap p_i|}{|g_i|}$, $PR_i = \frac{|g_i \cap p_i|}{|p_i|}$, $g_i = \{t^g_{i1}, t^g_{i2}, \ldots, t^g_{ik}\}$, $p_i = \{t^p_{i1}, t^p_{i2}, \ldots, t^p_{im}\}$. By using Planning Score, we aim to measure both the coverage of tools in each generated plan relative to the corresponding ground truth plan and the proportion of correct tools within the generated plan itself.

Execution Accuracy. We measure the percentage of correct outputs for each query and then calculate the average percentage across all queries to determine execution accuracy.

Response Accuracy. It measures the percentage of user queries that receive correct answers.

7.1.2 Baselines. We use two approaches with similar problem settings, HuggingGPT [49] and Sasha [46], as baselines. HuggingGPT uses LLMs to break down complex tasks into sub-tasks. It performs simple format checks, without considering the solvability of user queries or the grammar correctness of plans, and does not provide feedback to improve planning quality. Different from HuggingGPT, Sasha involves extra steps to prompt LLMs to evaluate the solvability of each user query and to improve the quality of the generated plan based on user feedback. To address identification issues in response generation for the baselines, we apply the same results formatting approach to them, enabling them to respond effectively. Besides, for a fair comparison, we provide Sasha with the same base information used by TaskSense's dynamic plan adaptation module, including data missing, data quality and execution results, for LLM-based plan regeneration.

LLMs. We use six popular LLMs as base models for the baselines and TaskSense: four commercial models (GPT-4 [5], GPT-4o [3], Claude-3 [22], and Claude-3.5 [22]), and two open-source models (Llama-3-70B [54] and Mistral [1]). Commercial ones are accessed via official APIs, while open-source models are accessed through AWS Bedrock. We set the temperature of LLMs to zero to minimize output randomness, with other settings left as default unless specified otherwise.

Table 3: Overall performance of different methods (using GPT-4 as the base LLM).

Dataset	Methods	Planning Accuracy	Execution Accuracy	Response Accuracy
In-lab	HuggingGPT	0.80	0.55	0.47
	Sasha	0.86	0.56	0.50
	Ours	**0.97**	**0.75**	**0.73**
DAHLIA	HuggingGPT	0.86	0.43	0.50
	Sasha	0.76	0.42	0.40
	Ours	**0.93**	**0.60**	**0.70**
Synthetic	HuggingGPT	0.66	0.59	0.50
	Sasha	0.32	0.67	0.48
	Ours	**0.96**	**0.72**	**0.74**
Real-world	HuggingGPT	0.55	0.35	0.55
	Sasha	0.66	0.37	0.62
	Ours	**0.82**	**0.64**	**0.75**

7.2 An End-to-End Application

To demonstrate TaskSense's workflow, we set up a prototype in a smart home environment. We deploy our system onto an NVIDIA Jetson AGX Orin [38] with 32 GB of GPU memory, connected to the internet to interact with LLM providers. We establish proof-of-concept testing with this setup, allowing users to query the system to acquire daily behavior insights with plans generated by TaskSense.

Figure 9 illustrates the plan generation, execution, and response steps based on the query posed by referring to the report of International Labor Organization [4]. From 12 p.m. to 4 p.m., the system dynamically switches between different execution paths for activity recognition based on environmental factors. For instance, when light levels drop below the RGB camera's threshold at 1 p.m., TaskSense switches to an alternative execution path using a depth camera, ensuring accurate behavior detection and correct response.

7.3 Overall Performance

The overall performance of TaskSense is evaluated by comparing its planning, execution, and response accuracy against baseline models across four datasets. Additionally, TaskSense's robustness is tested by comparing it with the baselines in six different LLM settings on the In-lab HAR dataset.

Table 3 shows evaluation results across different datasets using GPT-4, with TaskSense using hyperparameter $k = 10$ as the number of selected seed examples. The results indicate that TaskSense

Table 4: Overall performance on the In-lab dataset using different LLMs.

LLM	Methods	Planning Accuracy	Execution Accuracy	Response Accuracy
GPT-4o	HuggingGPT	0.83	0.51	0.60
	Sasha	0.83	0.47	0.63
	Ours	**0.97**	**0.72**	**0.70**
Claude-3-Opus	HuggingGPT	0.87	0.55	0.57
	Sasha	0.60	0.23	0.53
	Ours	**0.97**	**0.65**	**0.77**
Claude-3.5-Sonnet	HuggingGPT	0.73	0.54	0.50
	Sasha	0.83	0.48	0.73
	Ours	**0.93**	**0.67**	**0.70**
Llama 3 70B Instruct (Open-source)	HuggingGPT	0.47	0.27	0.37
	Sasha	0.63	0.48	0.37
	Ours	**0.77**	**0.54**	**0.57**
Mistral Large (Open-source)	HuggingGPT	0.73	0.58	0.47
	Sasha	0.40	0.18	0.30
	Ours	**0.97**	**0.74**	**0.70**

achieves up to 0.64 higher planning accuracy than the baselines. Most errors from TaskSense are due to solvable queries being treated as unsolvable, which can be mitigated with more precise queries. The planning accuracies of HuggingGPT are much lower than TaskSense because it does not conduct any check on the solvability and plan validity. Besides, the examples in its example library are randomly given, lacking sufficient diversity. Among its incorrect test samples, both cases of misjudging solvability and generating wrong plans account for large proportions. Sasha cannot achieve reliable planning performance as TaskSense, especially on the Synthetic dataset. It is because Sasha relies on LLMs for plan adjustments, but the comprehension abilities of LLMs are limited and their outputs involve a degree of randomness. Therefore, it is difficult for LLMs to make effective adjustments to the generated plans based on the complex feedback information from the external environment, resulting in incorrect or inaccurate adjustments. In contrast, the dynamic plan adoption mechanism of TaskSense uses a predefined module to adapt plans based on external information more accurately and promptly. With its dynamic plan adoption, TaskSense outperforms the baselines, improving execution accuracy by 0.29 and response accuracy by 0.30. Most of TaskSense's execution and response errors result from tool inaccuracies. Besides, tool inaccuracies cause inconsistency between the plan and execution accuracies for HuggingGPT and Sasha on the Synthetic dataset, where Sasha generates fewer correct plans but more tools with superior performance. TaskSense avoids this inconsistency as it always selects the optimal execution path for each plan.

Table 4 shows the results of TaskSense with five different LLMs. TaskSense consistently outperforms baselines, indicating its robustness and effectiveness across various LLMs. HuggingGPT tends to misclassify unsolvable queries as solvable (50% on average), while TaskSense handles these better with only a 4% error rate. Sasha struggles with solvable queries, showing a high error rate of 35%, while TaskSense has a lower error rate (10%). Unlike the baselines, TaskSense maintains consistent performance across LLMs. With GPT-4o, TaskSense performs best with 92.2% planning accuracy on average. With Claude-35-Sonnet, it handles solvable queries well but struggles with unsolvable ones. Due to Llama3's limited

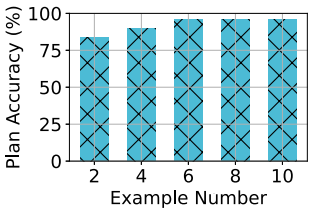

Figure 10: Planning accuracy under different numbers of retrieved seed examples.

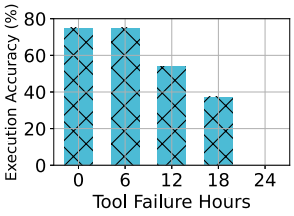

Figure 11: Execution accuracy under different levels of tool failure hours.

capabilities, TaskSense performs poorly with it, often confusing complex queries. The open-source model Mistral Large provides results comparable to commercial ones.

7.4 Microbenchmark

We examine 4 microbenchmarks: (1) the effect of retrieved seed example quantity during plan generation, (2) the impact of environmental factors' interference levels, (3) the comparison with RAG methods, and (4) the impact of the seed example library size.

7.4.1 Effect of Retrieved Seed Example Quantity. Figure 10 shows TaskSense's planning performance as the number of retrieved seed examples k varies on the In-lab dataset. The results indicate that its planning accuracy improves as k increases but converges when k reaches 6, after which more retrieved examples yield insignificant performance improvement.

7.4.2 Impact of Environmental Interference Levels. To evaluate the impact of environmental factors' interference levels, we use data with tool failures caused by environmental or device issues from the Synthetic dataset. Figure 11 shows TaskSense's execution performance as the tool failure hours through the day change. The results indicate that TaskSense with the design of dynamic plan adaptation effectively handles these changes and failures when the failure rate remains within a reasonable duration. For example, TaskSense can use a depth camera as an alternative during 8-hour RGB failures at night to maintain performance.

7.4.3 Comparison with RAG-based Approaches. TaskSense is further evaluated by comparing it with two RAG-based settings. The first setting, *planning example RAG*, applies RAG to the examples, which is developed based on TaskSense by removing grammar and query solvability checks and keeping only example retrieval during planning. The second setting, *tool output RAG*, applies RAG to inference results with the setting of continuously running all tools and storing their outputs in a results database. Under this setting, the system directly retrieves the corresponding tool execution outputs from the results database according to the user queries, while TaskSense allows users to choose which tools run continuously and which are invoked only on demand.

We compare our method with *planning example RAG* in terms of accuracy (Figure 12) and with *tool output RAG* in terms of system load (Figure 13) on the In-lab HAR dataset. The results indicate that TaskSense achieves up to a 19% higher planning score than *planning example RAG* across various numbers of retrieved examples. Besides, unlike *tool output RAG*'s fixed computational load (the blue and

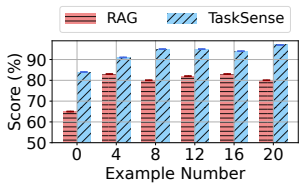

Figure 12: Comparison between planning example RAG and TaskSense in terms of planning score.

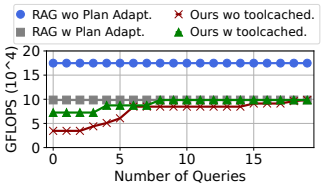

Figure 13: Comparison between tool output RAG and TaskSense in terms of cumulative GFLOPs.

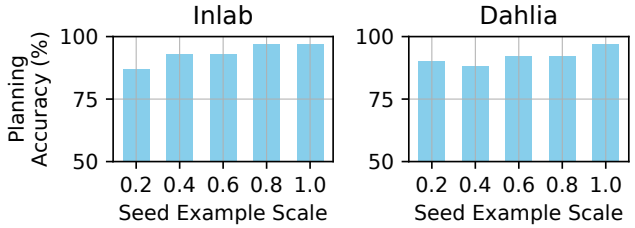

Figure 14: Impact of seed example scale on planning.

gray curves in Figure 13), the cumulative load of TaskSense 's load gradually increases as queries span more time ranges. The load of tool output RAG with dynamic plan adaptation (the gray curve) serves as an upper bound for TaskSense. Furthermore, the comparison between red and green curves indicates that more continuously running tools in TaskSense introduce a more fixed system overhead but reduce more response latency.

7.4.4 Impact of the Size of Seed Example Library. To assess the impact of seed example library size, we conduct experiments by varying the seed example library scale on the In-lab and Dahlia datasets. As shown in Figure 14, we consider the default setting of seed example scale used in §7.3 as '1.0', with '0.8' indicating the use of 80% of the seed examples. The results show that TaskSense's planning performance declines as the seed example scale decreases, indicating the importance of seed examples to optimize TaskSense's planning ability.

7.5 Ablation Study

Plan Generation. Table 5 shows the planning performance of TaskSense and its variants with module ablations, using GPT-4 as the base LLM on the Synthetic dataset. The results indicate that all modules contribute to task planning. Among these modules, the example library contributes the most to both planning accuracy and score (0.28 and 0.23, respectively). Moreover, when the negative examples are removed, our evaluation shows that the planning accuracy and score drop to 0.92 and 0.95, respectively. The drops are mainly due to incorrect tool selection, indicating the original LLM's limited understanding of the tool capability boundaries.

The creativity of LLMs also impacts the quality of plans. The above experiments are conducted with a temperature setting of 0. When the creativity of GPT-4 increases (temperature = 1.5), the accuracy and score decrease to 0.74 and 0.80, respectively. In these cases, plan verification becomes helpful as a remedy, improving accuracy to 0.80 and score to 0.87.

Table 5: Impact of components on planning performance.

Methods	Planning Acc.	Planning Score
w/o Sensor Lang. Instruction	0.88	0.91
w/o Example Library	0.58	0.76
w/o Checking	0.88	0.91
TaskSense	**0.96**	**0.99**

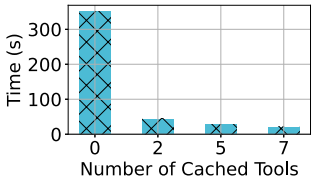

Figure 15: Impact of cache module on overall execution latency.

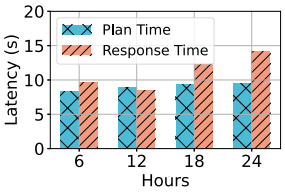

Figure 16: Impact of sensor data time range on latencies during planning and responding stages.

Dynamic Plan Adaptation. The effectiveness of the plan adaptation design is evaluated on the In-lab dataset. TaskSense with the dynamic plan policy achieves the highest execution accuracy at 0.82, while methods using fixed plan policies with RGB and depth modalities achieve lower accuracies of 0.56 and 0.60, respectively. This demonstrates the effectiveness of dynamic plan adaptation in enhancing execution accuracy compared to using fixed plan paths.
Results Formatting. The impact of results formatting on response accuracy is evaluated using the In-lab dataset. With formatting, response accuracy increases to 73.3%, compared to 66.7% without it. This shows that results formatting effectively enhances the accuracy of the system's responses.

7.6 System Overhead

In this section, we evaluate the system overhead of TaskSense, including execution latency and the API calling cost of LLMs.
Execution Latency. We evaluate the impact of the cache module on overall latency using the In-lab dataset. Figure 15 shows that the cache module significantly reduces the execution latency of the system. Across varying numbers of cached tools, the total time drops from 360 to 31 seconds, driven primarily by the decline in plan execution time from 342 to 11 seconds. Meanwhile, plan generation and response generation latencies remain stable, fluctuating between 8-10 seconds and 10-12 seconds, respectively. Besides, Figure 16 shows the planning and response times across different time scales: 6, 12, 18, and 24 hours. The results indicate that as data amount increases, both planning and response times rise accordingly.
LLM API Calling Cost. We evaluate the LLM API calling cost of TaskSense on the In-lab dataset. Table 6 shows the detailed breakdown of the token cost of each component in TaskSense. The total fixed token count for goal descriptions in the planning and responding stages of our approach is 745, which is comparable to HuggingGPT (433) and Sasha (1069). This similarity in fixed token usage does not result in a significant cost difference. For the variable parts, TaskSense includes additional components, such as grammar

Table 6: Breakdown of LLM (GPT-4) token counts.

Stage	Module	# Avg. Token Used
Planning	1. Goal Description	558
	2. Examples	129/example
	3. Vocabulary Set	137/tool
	4. Grammar Rules	11/tool
Responding	1. Goal Description	187
	2. Execution Results	63/time interval

and examples, which are not present in HuggingGPT and Sasha. Specifically, HuggingGPT lacks grammar, while Sasha lacks both examples and grammar. On average, each additional grammar rule increases token usage by 11 tokens, and each example adds 129 tokens. Despite these increases, the overall cost of our method does not rise significantly and remains acceptable, making our system cost-effective for practical applications.

8 DISCUSSION

Scalability to FM tools. The current toolset only contains task-specific AI models. Many multi-modal foundation models, such as LLaVA [34] and ImageBind [19], can perform diverse downstream tasks with one model. In future work, TaskSense could be enhanced to support more complex tool dependencies and expanded to integrate these multi-modal models into the toolset, enabling it to handle broader tasks.

Dynamic Plan Adaptation Strategy. TaskSense offers strong compatibility, and its dynamic plan adaptation strategy can be customized. Currently, TaskSense mainly focuses on the quality of sensor data and execution results in real-world scenarios using a strategy combining pre-execution filtering and post-execution selection. Future work could extend TaskSense for additional needs by implementing other strategies, such as path selection based on execution latency for real-time performance or using multimodal fusion with all available data streams to enhance downstream task performance [39].

Reducing LLM API calling cost. At present, TaskSense needs to call LLM APIs for each new query, resulting in unavoidable financial expenses and time overheads. Future work could introduce a cache mechanism to avoid frequent API calls for similar queries and a more compact representation of the plan execution results.

9 CONCLUSION

This paper studies utilizing LLMs to coordinate sensor systems for complicated user queries. To achieve this goal, this paper proposes a translation-like approach based on a Sensor Language definition to interpret human intents to executable plans. To improve execution robustness, TaskSense adopts a dynamic plan adaptation mechanism, adapting plans based on feedback from environmental factors. We prototyped TaskSense in an end-to-end application setting and tested it with four datasets. Evaluation results show that TaskSense offers up to 3× planning accuracy and 1.75× responding accuracy compared with baselines.

ACKNOWLEDGEMENT

This paper is supported in part by the National Natural Science Foundation of China (NSFC) under 62202407 and the Research Grants Council (RGC) of Hong Kong under GRF 14214022, GRF 14212323 and TRS T43-513/23-N.

REFERENCES

[1] 2023. Mistral Inference. https://github.com/mistralai/mistral-inference,.
[2] 2024. Autonomous & Sensor Technology. https://www.statista.com/outlook/tmo/artificial-intelligence/autonomous-sensor-technology/worldwide,.
[3] 2024. Hello GPT-4o. https://openai.com/index/hello-gpt-4o/,.
[4] 2024. International Labour Organization. https://www.ilo.org/,.
[5] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).
[6] Alfred V Aho, John E Hopcroft, and Jeffrey D Ullman. 1973. On finding lowest common ancestors in trees. In *Proceedings of the fifth annual ACM symposium on Theory of computing*. 253–265.
[7] Cohere AI. 2023. Introduction to Text Embeddings. https://github.com/cohere-ai/notebooks/blob/main/notebooks/llmu/Introduction_Text_Embeddings.ipynb?ref=cohere-ai.ghost.io.
[8] Amazon Web Services. 2024. Amazon Bedrock: Build Generative AI Applications with Foundation Models. https://aws.amazon.com/bedrock/. Accessed: 2024-06-30.
[9] Home Assistance. 2024. Home Assistance: Awaken Your Home. https://www.home-assistant.io/.
[10] Daniel J Buysse, Martica L Hall, Patrick J Strollo, Thomas W Kamarck, Jane Owens, Laisze Lee, Steven E Reis, and Karen A Matthews. 2008. Relationships between the Pittsburgh Sleep Quality Index (PSQI), Epworth Sleepiness Scale (ESS), and clinical/polysomnographic measures in a community sample. *Journal of clinical sleep medicine* 4, 6 (2008), 563–571.
[11] Huimin Chen, Chaojie Gu, Lilin Xu, Rui Tan, Shibo He, and Jiming Chen. 2025. Listen to your face: a face authentication scheme based on acoustic signals. *ACM Transactions on Sensor Networks* 21, 1 (2025), 1–23.
[12] Hao Chen, Yiming Zhang, Qi Zhang, Hantao Yang, Xiaomeng Hu, Xuetao Ma, Yifan Yanggong, and Junbo Zhao. 2023. Maybe only 0.5% data is needed: A preliminary exploration of low training data instruction tuning. *arXiv preprint arXiv:2305.09246* (2023).
[13] C Craig, A Marshall, M Sjostrom, A Bauman, P Lee, D Macfarlane, T Lam, and S Stewart. 2017. International physical activity questionnaire-short form. *J Am Coll Health* 65, 7 (2017), 492–501.
[14] Hongwei Cui, Yuyang Du, Qun Yang, Yulin Shao, and Soung Chang Liew. 2023. Llmind: Orchestrating ai and iot with llms for complex task execution. *arXiv preprint arXiv:2312.09007* (2023).
[15] Ryandito Diandaru, Lucky Susanto, Zilu Tang, Ayu Purwarianti, and Derry Wijaya. 2024. What Linguistic Features and Languages are Important in LLM Translation? *arXiv preprint arXiv:2402.13917* (2024).
[16] Hugging Face. 2024. FFmpeg: A complete, cross-platform solution to record, convert and stream audio and video. https://www.ffmpeg.org/.
[17] Hugging Face. 2024. Hugging Face Community. https://huggingface.co/.
[18] Jacopo Galli, D Meucci, Giampiero Salonna, R Anzivino, Valentina Giorgio, M Trozzi, Stefano Settimi, ML Tropiano, Gaetano Paludetti, and S Bottero. 2020. Use OF NBI for the assessment of clinical signs of rhino-pharyngo-laryngeal reflux in pediatric age: Preliminary results. *International Journal of Pediatric Otorhinolaryngology* 128 (2020), 109733.
[19] Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra. 2023. ImageBind: One Embedding Space To Bind Them All. In *CVPR*.
[20] Christopher G Goetz, Barbara C Tilley, Stephanie R Shaftman, Glenn T Stebbins, Stanley Fahn, Pablo Martinez-Martin, Werner Poewe, Cristina Sampaio, Matthew B Stern, Richard Dodel, et al. 2008. Movement Disorder Society-sponsored revision of the Unified Parkinson's Disease Rating Scale (MDS-UPDRS): scale presentation and clinimetric testing results. *Movement disorders: official journal of the Movement Disorder Society* 23, 15 (2008), 2129–2170.
[21] Jiaming Han, Kaixiong Gong, Yiyuan Zhang, Jiaqi Wang, Kaipeng Zhang, Dahua Lin, Yu Qiao, Peng Gao, and Xiangyu Yue. 2024. Onellm: One framework to align all modalities with language. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 26584–26595.
[22] Jussi S Jauhiainen and Agustín Garagorry Guerra. 2024. Evaluating Students' Open-ended Written Responses with LLMs: Using the RAG Framework for GPT-3.5, GPT-4, Claude-3, and Mistral-Large. *arXiv preprint arXiv:2405.05444* (2024).
[23] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C Park. 2024. Database-Augmented Query Representation for Information Retrieval. *arXiv preprint arXiv:2406.16013* (2024).
[24] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. 2017. Google's multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics* 5

(2017), 339–351.
[25] Evan King, Haoxiang Yu, Sangsu Lee, and Christine Julien. 2024. Sasha: creative goal-oriented reasoning in smart homes with large language models. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 8, 1 (2024), 1–38.
[26] Philipp Koehn. 2009. *Statistical machine translation*. Cambridge University Press.
[27] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems* 35 (2022), 22199–22213.
[28] Roman Koshkin, Katsuhito Sudoh, and Satoshi Nakamura. 2024. Transllama: Llm-based simultaneous translation system. *arXiv preprint arXiv:2402.04636* (2024).
[29] Labelbox. 2024. GPT-3.5: Models - OpenAI. https://labelbox.com/product/model/foundry-models/gpt-3-5/. Accessed: 2024-06-30.
[30] Guillaume Lample and Alexis Conneau. 2019. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291* (2019).
[31] McMahon E Lawton. 2008. Brody instrumental activities of daily living scale (IADL). *MaineHealth* 108, 4 (2008), 2.
[32] Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Luke Zettlemoyer, Omer Levy, Jason Weston, and Mike Lewis. 2023. Self-alignment with instruction backtranslation. *arXiv preprint arXiv:2308.06259* (2023).
[33] Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. 2023. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *arXiv preprint arXiv:2303.16434* (2023).
[34] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual Instruction Tuning. In *NeurIPS*.
[35] Zhaoyang Liu, Zeqiang Lai, Zhangwei Gao, Erfei Cui, Zhiheng Li, Xizhou Zhu, Lewei Lu, Qifeng Chen, Yu Qiao, Jifeng Dai, et al. 2023. Controlllm: Augment language models with tools by searching on graphs. *arXiv preprint arXiv:2310.17796* (2023).
[36] Ning Miao, Yee Whye Teh, and Tom Rainforth. 2023. Selfcheck: Using llms to zero-shot check their own step-by-step reasoning. *arXiv preprint arXiv:2308.00436* (2023).
[37] Microsoft. 2024. Azure OpenAI Service. https://azure.microsoft.com/en-us/products/ai-services/openai-service.
[38] NVIDIA Corporation. 2023. Jetson Orin Modules and Developer Kits. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/
[39] Xiaomin Ouyang, Xian Shuai, Yang Li, Li Pan, Xifan Zhang, Heming Fu, Sitong Cheng, Xinyan Wang, Shihua Cao, Jiang Xin, et al. 2024. ADMarker: A Multi-Modal Federated Learning System for Monitoring Digital Biomarkers of Alzheimer's Disease. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. 404–419.
[40] Xiaomin Ouyang, Xian Shuai, Yang Li, Li Pan, Xifan Zhang, Heming Fu, Xinyan Wang, Shihua Cao, Jiang Xin, Hazel Mok, et al. 2023. ADMarker: A Multi-Modal Federated Learning System for Monitoring Digital Biomarkers of Alzheimer's Disease. *arXiv preprint arXiv:2310.15301* (2023).
[41] Xiaomin Ouyang and Mani Srivastava. 2024. LLMSense: Harnessing LLMs for High-level Reasoning Over Spatiotemporal Sensor Traces. *arXiv preprint arXiv:2403.19857* (2024).
[42] Haojie Pan, Zepeng Zhai, Hao Yuan, Yaojia Lv, Ruiji Fu, Ming Liu, Zhongyuan Wang, and Bing Qin. 2023. Kwaiagents: Generalized information-seeking agent system with large language models. *arXiv preprint arXiv:2312.04889* (2023).
[43] Poe. 2024. Poe Platform. https://poe.com/.
[44] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789* (2023).
[45] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2024. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems* 36 (2024).
[46] Dhruv Shah, Błażej Osiński, Sergey Levine, et al. 2023. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. In *Conference on robot learning*. PMLR, 492–504.
[47] Azmeh Shahid, Kate Wilkinson, Shai Marcu, and Colin M Shapiro. 2011. Fatigue assessment scale (FAS). In *STOP, THAT and one hundred other sleep scales*. Springer, 161–162.
[48] Yifei Shen, Jiawei Shao, Xinjie Zhang, Zehong Lin, Hao Pan, Dongsheng Li, Jun Zhang, and Khaled B Letaief. 2024. Large language models empowered autonomous edge ai for connected intelligence. *IEEE Communications Magazine* (2024).
[49] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2024. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems* 36 (2024).
[50] Shuyao Shi, Neiwen Ling, Zhehao Jiang, Xuan Huang, Yuze He, Xiaoguang Zhao, Bufang Yang, Chen Bian, Jingfei Xia, Zhenyu Yan, et al. 2024. Soar: Design and Deployment of A Smart Roadside Infrastructure System for Autonomous Driving. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. 139–154.
[51] Amit Kumar Sikder, Leonardo Babun, Z Berkay Celik, Hidayet Aksu, Patrick McDaniel, Engin Kirda, and A Selcuk Uluagac. 2022. Who's controlling my device? Multi-user multi-device-aware access control system for shared smart home environment. *ACM Transactions on Internet of Things* 3, 4 (2022), 1–39.
[52] Spotify. 2024. Approximate Nearest Neighbors Oh Yeah. https://github.com/spotify/annoy.
[53] Jennifer Yee-man Tang, Andy Hau-yan Ho, Hao Luo, Gloria Hoi-yan Wong, Bobo Hi-po Lau, Terry Yat-sang Lum, and Karen Siu-lan Cheung. 2016. Validating a Cantonese short version of the Zarit Burden Interview (CZBI-Short) for dementia caregivers. *Aging & Mental Health* 20, 9 (2016), 996–1001.
[54] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).
[55] TP-link. 2024. TP-link Smart Home Community. https://community.tp-link.com/en/smart-home/.
[56] Geoffrey Vaquette, Astrid Orcesi, Laurent Lucat, and Catherine Achard. 2017. The daily home life activity dataset: a high semantic activity dataset for online recognition. In *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*. IEEE, 497–504.
[57] Vzense. 2024. Vzense Technology. https://www.vzense.com/.
[58] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291* (2023).
[59] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244* (2023).
[60] Lilin Xu, Chaojie Gu, Rui Tan, Shibo He, and Jiming Chen. 2023. MESEN: Exploit Multimodal Data to Design Unimodal Human Activity Recognition with Few Labels. In *Proceedings of the 21st ACM Conference on Embedded Networked Sensor Systems*. 1–14.
[61] Lilin Xu, Keyi Wang, Chaojie Gu, Xiuzhen Guo, Shibo He, and Jiming Chen. 2024. GesturePrint: Enabling user identification for mmWave-based gesture recognition systems. In *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1074–1085.
[62] Bufang Yang, Yunqi Guo, Lilin Xu, Zhenyu Yan, Hongkai Chen, Guoliang Xing, and Xiaofan Jiang. 2024. SocialMind: LLM-based Proactive AR Social Assistive System with Human-like Perception for In-situ Live Interactions. *arXiv preprint arXiv:2412.04036* (2024).
[63] Bufang Yang, Lixing He, Neiwen Ling, Zhenyu Yan, Guoliang Xing, Xian Shuai, Xiaozhe Ren, and Xin Jiang. 2023. Edgefm: Leveraging foundation model for open-set learning on the edge. In *Proceedings of the 21st ACM Conference on Embedded Networked Sensor Systems*. 111–124.
[64] Bufang Yang, Lixing He, Kaiwei Liu, and Zhenyu Yan. 2024. VIAssist: Adapting Multi-modal Large Language Models for Users with Visual Impairments. *arXiv preprint arXiv:2404.02508* (2024).
[65] Bufang Yang, Siyang Jiang, Lilin Xu, Kaiwei Liu, Hai Li, Guoliang Xing, Hongkai Chen, Xiaofan Jiang, and Zhenyu Yan. 2024. Drhouse: An llm-empowered diagnostic reasoning system through harnessing outcomes from sensor data and expert knowledge. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 8, 4 (2024), 1–29.
[66] Bufang Yang, Le Liu, Wenxuan Wu, Mengliang Zhou, Hongxing Liu, and Xinbao Ning. 2023. BrainZ-BP: A Non-invasive Cuff-less Blood Pressure Estimation Approach Leveraging Brain Bio-impedance and Electrocardiogram. *IEEE Transactions on Instrumentation and Measurement* (2023).
[67] Huanqi Yang, Sijie Ji, Rucheng Wu, and Weitao Xu. 2024. Are You Being Tracked? Discover the Power of Zero-Shot Trajectory Tracing with LLMs! *arXiv preprint arXiv:2403.06201* (2024).
[68] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629* (2022).
[69] Jinliang Yuan, Chen Yang, Dongqi Cai, Shihe Wang, Xin Yuan, Zeling Zhang, Xiang Li, Dingge Zhang, Hanzi Mei, Xianqing Jia, et al. 2024. Mobile Foundation Model as Firmware. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. 279–295.
[70] Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. 2023. Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406* (2023).
[71] Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2024. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems* 36 (2024).
[72] Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Zhicheng Dou, and Ji-Rong Wen. 2023. Large language models for information retrieval: A survey. *arXiv preprint arXiv:2308.07107* (2023).