Heuristically Motivating Large Language Models for Task Planning

Chao Wang[a,b,*], Longhui Cao[a,b], Juntong Qi[a,b] and Linqi Ye[a,b,*]

[a]School of Future Technology, Shanghai University, Shanghai, 200444, China
[b]Institute of Artificial Intelligence, Shanghai University, Shanghai, 200444, China

ARTICLE INFO

Keywords:
Autonomous Agents
Decision Making
Embodied-AI
Large Language Model
Tasks Planning
Information Extraction

ABSTRACT

Recently, leveraging Large Language Models (LLMs) as planners has emerged as a mainstream approach for complex planning tasks. However, as it remains unclear whether LLMs' performance in these tasks stems from 'memory' or 'logic', researchers are divided on their inherent planning capabilities. Recognizing the widely acknowledged commonsense knowledge embedded within LLMs, we propose a novel planning framework, **HSP**, which integrates `Heuristic`, `Supplementary`, and `Plan-grounding` modules. In the HSP framework, LLMs do not function as the primary planners; instead, they serve as reasoning and refinement engines within the `Supplementary` and `Plan-grounding` modules. This approach capitalizes on the text generation capabilities of LLMs to both rationalize heuristic plans and correct erroneous actions during execution. Specifically, the `Heuristic` module generates initial planning paths based on the Planning Domain Description Language (PDDL). The `Supplementary` module then employs LLMs to refine and enrich these paths, while the `Plan-grounding` module utilizes LLMs to filter out non-executable operations, ensuring a seamless transition from abstract planning to embodied grounding. Empirical evaluations across 10 household task domains demonstrate that HSP significantly outperforms existing state-of-the-art frameworks. Notably, HSP excels in long-horizon tasks; for example, in multi-step scenarios such as *'Bring mug and cupcake to the coffee table'*, HSP achieves a 100% success rate, nearly doubling the 54% of the optimal baseline. Furthermore, HSP consistently completes most tasks with the minimum number of steps, proving its superior efficiency and reliability.

1. Introduction

One of the unique features of human intelligence is the ability to solve complex problems. Through independent reasoning, humans can devise viable, logical long-term plans [1]. Inspired by this, embodied intelligence—a paradigm integrating AI with physical entities (e.g., robots)—endows agents with the ability to perceive, learn, and interact with complex environments, enabling them to generate and execute plans for complex tasks [2, 3]. Embodied intelligence offers diverse approaches to planning problems. In classical planning methods, long-term task planning for agents is often modeled as a joint symbolic and geometric reasoning problem [4, 5, 6, 7]. This approach enables agents to build an autonomous decision-making intelligent system capable of solving complex tasks. However, these mathematically transformed planning methods strip away the warmth of language, making agents less accessible. After all, what we consider 'intelligence' is deeply tied to natural language interaction [8].

Fortunately, recent research on Large Language Models (LLMs) brings exciting news. On one hand, providing LLMs with appropriate cues helps them perform tasks such as arithmetic, knowledge acquisition, and reasoning [9, 10, 11, 12, 13]. On the other hand, several studies apply the common-sense knowledge acquired by LLMs to planning tasks with exciting results [14, 15, 16]. To enable natural language-mediated planning in embodied intelligence, significant research efforts focus on integrating LLMs into autonomous agent architectures. Previous efforts advocate LLMs as principal planners, capitalizing on their planning generation competencies [14, 8, 17]. In existing research, LLMs are primarily applied to planning and correction modules, utilizing the models' extensive commonsense knowledge to solve planning problems. Nevertheless, the prompt sensitivity of LLMs leads to instability in both planning performance and correction frequency. In contrast, emerging research explores employing LLMs in auxiliary planning capacities rather than primary control roles, demonstrating their effectiveness as decision-support modules for agent planning [18, 19].

The role of LLMs in task planning remains debated, primarily due to conflicting views on their intrinsic planning capabilities. Some studies [20, 21] argue that LLMs lack inherent planning capabilities. Valmeekam et al. [22] assess

*Corresponding authors
cwang@shu.edu.cn (C. Wang); caolojghui@shu.edu.cn (L. Cao); qijt@shu.edu.cn (J. Qi); yelinqi@shu.edu.cn (L. Ye)
ORCID(s):

the planning ability of LLMs on a range of planning problems based on the type of domains used in the International Planning Competition [23]. The experimental results show that even for the strongest LLM, GPT-4, only about 14% of generated plans are correct, and they also show that fine-tuning does not significantly improve this result. However, some studies consider LLMs to be planning-capable [24, 25]. The work of Yao et al. [8] affirms the planning potential of LLMs. They believe that even if LLMs cannot generate plans at once, the accuracy of plan generation is able to improve through iteration, and their best trial in Alfworld [26] achieves a success rate of 71%. In contrast, Stechly et al. [27] refuted this idea through a validation experiment. Their results show that there is no significant improvement compared to the baseline, either by directly generating solutions or by criticizing the generated solutions using LLMs. Since LLMs are large neural networks trained on massive amounts of data, the exact composition of their training data is unknown, and therefore the provenance of LLM-derived task planning is unverifiable. This leads to instability in task planning using LLMs. *It is difficult to expect planning methods from unknown sources to perform reliably on complex tasks.*

For instance, consider a typical household task such as "putting an apple into the refrigerator". An agent that relies primarily on an LLM as its planner may produce unstable or even erroneous plans due to subtle variations in prompts or inherent uncertainties in its internal knowledge. The agent might incorrectly generate an initial step like "walk to the sofa"—a common but irrelevant action—or omit a critical step such as "open the refrigerator door." Even when iterative feedback is applied for correction, the final plan generated may still contain redundant loops (e.g., repeatedly opening and closing the refrigerator door) or actions that are infeasible in the given context (e.g., attempting to pick up a "fruit bowl" that is not present in the scene). This instability and unreliability pose significant challenges for pure LLM-based planning methods when applied to complex, long-horizon tasks that demand precise and dependable sequential reasoning.

Although numerous recent studies employ LLMs alone for home task planning, their inherent limitations—unverifiable reasoning processes, prompt sensitivity, and lack of formal guarantees—persist as fundamental obstacles in complex scenarios. The Planning Domain Description Language (PDDL), in contrast, provides a formally verifiable foundation for task planning. Its unique contributions are threefold: (1) *Stability through logical verification*: PDDL-based planners guarantee that generated plans are logically consistent and executable within the defined state transition system; (2) *Minimal and interpretable planning skeletons*: PDDL produces concise, non-redundant action sequences that serve as reliable blueprints for subsequent refinement; (3) *Explicit modeling of long-range dependencies*: By formally specifying preconditions and effects, PDDL explicitly encodes state dependencies that LLMs often overlook or hallucinate. Therefore, we employ PDDL not as a replacement for LLMs, but as a stabilizing core that ensures basic plan correctness. This stable foundation then enables the secure integration of LLMs, allowing them to contribute their unique strengths in commonsense reasoning, natural language understanding, and adaptive refinement without compromising the plan's logical integrity. This PDDL+LLM hybrid approach fundamentally differs from and outperforms mere prompt-engineering with LLMs, as it systematically combines the verifiable rigor of classical planning with the flexible generalization of LLMs. Consequently, LLMs are not tasked with the error-prone process of generating a plan from scratch, but are leveraged to enhance and ground an already-correct plan skeleton.

Luckily, the excellent text generation capabilities that LLMs have are indeed real [28, 29, 30, 31]. Our framework capitalizes on this characteristic of LLMs, thereby equipping intelligent agents with natural language interaction capabilities. Simultaneously, to *mitigate the instability of the planning capacity associated with LLMs*, we propose a novel planning framework with the `Heuristic` module, `Supplementary` module, and `Plan-grounding` module, and denoted it as HSP.

The first module in our framework, designated as `Heuristic`, integrates the stability of classical planning approaches to derive preliminary heuristic schemes using the Planning Domain Description Language (PDDL). As shown in Figure 1, the overall workflow of the HSP is illustrated. Unlike approaches that use LLMs to generate domain/problem files [32], our PDDL files exhibit higher accuracy. After obtaining the heuristic plan, we refine it using the second module, `Supplementary`. In this module, the LLMs generate the actions and objects for each specific step based on the content of the heuristic planning. However, the feasibility of the plan after the `Supplementary` module cannot be guaranteed, and the resulting plan may contain duplicate actions as well as actions that cannot be executed by the agent. Therefore, the planning is then handed over to the third module, `Plan-grounding`, which operates both before action execution and during execution. At the beginning of each action, it is used to remove duplicates and convert infeasible actions into feasible ones. During execution, the `Plan-grounding` module is responsible for analyzing the current state and making decisions about the next state.

To validate the effectiveness of the HSP framework, we select the challenging Virtual Home benchmark [33]. A large number of everyday items required for home tasks are included in this benchmark. As a recognized benchmark

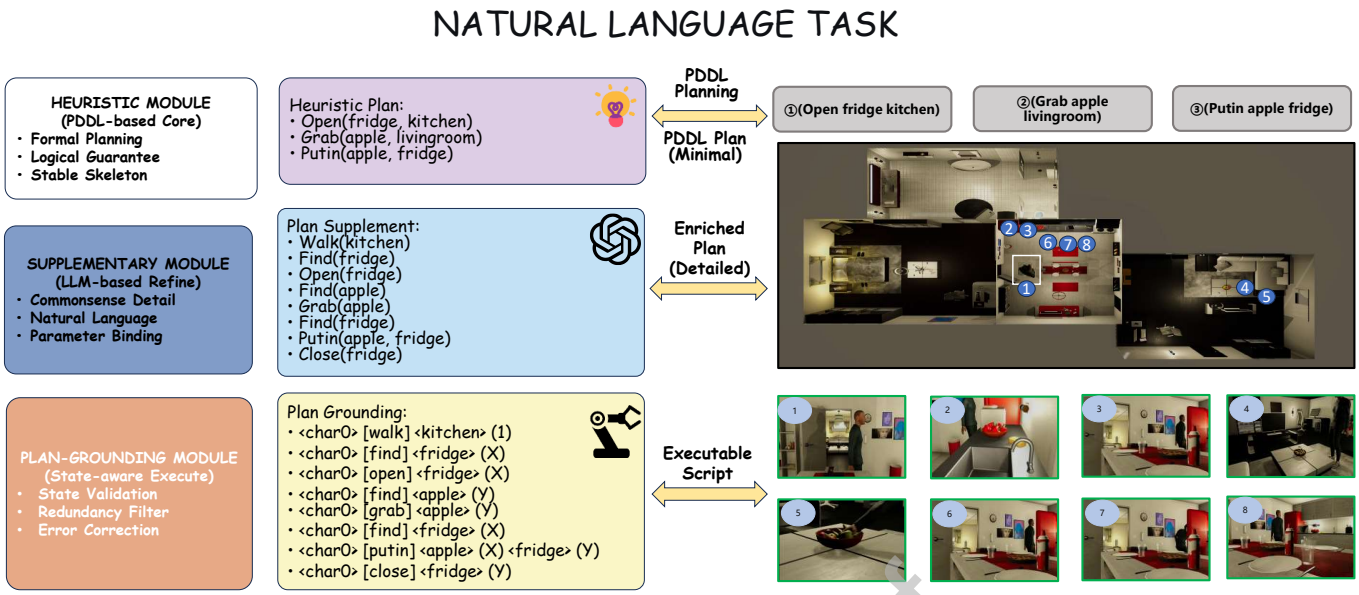

Figure 1: HSP Framework Overview: A Three-Stage Planning Pipeline. This framework transforms natural language tasks into executable scripts through three specialized modules. The left side illustrates the abstraction processing stage, while the right side demonstrates the data transformation process across each stage using a concrete example: *'Put Apple in Fridge'*.

for home tasks, it provides a suitable scenario for agents to be tested in a simulation environment [34, 33]. We select 10 challenging daily tasks from Virtual Home and compare our framework with popular baseline methods, which include ProgPrompt [35] and HiCRISP [36]. We experimentally demonstrate that our framework performs best on these tasks. Baseline frameworks fail to complete certain tasks, whereas our HSP framework successfully accomplishes all of them. Our framework significantly outperforms in the planning of long tasks. For the task "Bring mug and cupcake to the coffee table", our framework improves the success rate from 54% to 100%, achieving a 2× improvement over the best-performing baseline, HiCRISP [36]. Furthermore, our framework completes most tasks using **the fewest** number of steps. Figure 2 shows the advantages of our framework over other frameworks. Overall, our main **contributions** are summarized as follows:

1. *A novel planning framework, HSP.* Through HSP, agents are empowered to plan for complex tasks.
2. *A new way of thinking about the combination of LLMs and task planning.* HSP's planning capability stems primarily from classical methods, not LLMs. By augmenting classical planning with LLMs, this hybrid approach significantly improves generalization.
3. *Extensive experiments demonstrate the significant advantages of our framework on challenging home tasks.* It is demonstrated by experimental results that our HSP not only outperforms other baseline frameworks in terms of success rate but also completes the task with minimal step counts.

The rest of the paper is structured as follows: Section 2 presents different approaches to task planning for intelligent agents, as will as various perspectives on whether LLMs can be utilized for planning. Section 3 provides the necessary notation and identifies the tasks addressed in this study. Section 4 elaborates on the main module of our framework, HSP, along with its technical details. Experimental results are presented in Section 5. Section 6 summarizes the contributions of this paper and explores the limitations of our framework. Finally, Section 7 concludes the study and analyzes how future work addresses the current limitations.

2. Related Work

This section first explores the utilization of classical planning algorithms for solving agent planning tasks. Subsequently, following the advancement of LLMs, it analyzes the applicability of LLMs in complex task planning.

Figure 2: Application of HSP in planning. Taking the task *'Put apple in fridge'* as an example, we see that the baseline ProgPrompt is able to accomplish the task, but uses a lot of repetitive steps that add to the complexity of the task. HiCRISP, on the other hand, performs additional steps after completing the task, causing it to fail. Among these planning frameworks, only our framework HSP is able to successfully complete the task with the least number of steps.

Additionally, we focus on synthesizing recent research that integrates LLMs with PDDL-based planning, highlighting the methodological distinctions between these approaches and our proposed HSP framework. Finally, we summarize the different viewpoints in the academic community regarding the feasibility of using LLMs for task planning.

2.1. Classical Planning

Planning Domain Description Language (PDDL). The PDDL serves as the foundational action language for classical task planning systems. Traditional PDDL-based planners typically employ heuristic search strategies to derive feasible action sequences [37, 38, 39]. This paradigm requires explicit environment modeling through two key components. First, a domain file specifies action schemas with their preconditions and effects, along with relevant predicate relationships. Second, it is necessary to obtain a problem file defining initial and goal states.

Recent advances have extended PDDL's capabilities for complex task execution. Garrett et al. [40] developed a domain-independent algorithm that transforms PDDL Stream problems into sequential PDDL subproblems, augmented with greedy optimization for constraint-satisfaction scenarios. Through systematic empirical analysis, Jiang et al. [41] demonstrate that PDDL outperforms Answer Set Programming (ASP) in long-horizon planning contexts, whereas ASP has been shown to be more effective for complex reasoning tasks involving extensive large-scale object types.

While classical PDDL approaches have been remarkably successful, they remain fundamentally constrained by several long-standing issues. In higher-complexity environments, the exponential growth of action-object interactions renders this method computationally intractable. In addition, models generated by traditional planning methods lack extensibility, making it necessary to re-model in every new application environment.

Compared to PDDL methods, by leveraging LLMs' embedded world knowledge, HSP eliminates the need for fixed-task constraints. Furthermore, HSP's LLM-assisted PDDL file generation minimizes manual specification requirements and enhances environment modeling efficiency.

Reinforcement learning (RL). Compared to the PDDL approach for planning tasks, the learning-based method relaxes constraints [42, 4, 5, 43]. Numerous studies demonstrate the feasibility of using reinforcement learning for task planning [44, 45, 46]. Eysenbach et al. [7] highlight that sparse rewards in long-term planning hinder the effectiveness of reinforcement learning. To address this, they propose the Search on Replay Buffer (SoRB) method. By using goal-conditioned reinforcement learning to find the most suitable midpoint signposts, the long-term task is decomposed, allowing the agent to solve sparse-reward tasks within hundreds of steps while improving generalization. Xu et al. [6] propose the Neural Task Programming (NTP) machine learning framework, in which the ideas of few-shot learning and neural program induction are successfully combined. Specifically, NTP takes a task description as input and recursively decomposes it into finer subtasks. Afterward, these subtasks are fed into a hierarchical neural network for processing. Finally, the task is accomplished through a callable underlying program that interacts with the environment. Xu et al. [47] introduce a regression planning network by drawing on regression planning ideas. This

network initiates backward planning from the target task and generates a series of intermediate goals to reach the initial state. Experiments demonstrate that the method generalizes and learns from visual inputs in an end-to-end manner. Shah et al. [48] analyze the value function corresponding to each low-level skill in a long-term task, and based on this, propose the concept of 'value function space'. By utilizing the 'value function space', the noise interference in the long-term task is able to be removed. This method improves the performance of long-term task planning and better achieves zero-shot generalization. Despite constraint relaxation, learning-based methods exhibit sensitivity to data distribution shifts and require stronger generalization.

Undeniably, the aforementioned planning methods confer partial planning capabilities to agents. Specifically, classical planning algorithms exhibit higher stability in well-defined tasks, albeit at the cost of extensive manual configuration. On the one hand, PDDL-based planning necessitates prior environment modeling, demanding non-trivial PDDL expertise. On the other hand, learning-based approaches inherently depend on neural network training, where competent planning performance mandates substantial training datasets. Our HSP framework is grounded in the stability of classical planning but seeks to mitigate its manual burden and rigidity through a novel integration with LLMs.

Our approach is also based on the paradigm of neural network generation. The key distinction lies in the computational overhead: whereas conventional methods demand substantial time to build task-specific networks, our approach leverages pre-trained LLMs without fine-tuning requirements.

2.2. Planning With LLMs

The emergence of LLMs suggests a new research direction for task planning. Leveraging their advanced text generation capabilities, LLMs enable agents to accomplish complex task planning[17, 34, 49, 50, 51, 52].

Unlike the learning-based approaches mentioned above, LLMs tend to solve more planning tasks due to their extensive training data. An experimental study by Huang et al. [33] demonstrates that sufficiently large pre-trained LLMs, when provided with suitable prompts, can effectively decompose long-term tasks into short-term tasks without further training. However, planning solutions obtained directly from LLMs exhibit significant instability, with performance varying significantly across different prompts. Silver et al. [53] investigate the integration of LLMs with PDDLs. In their experiments, they find that the PDDL planning problem can be effectively solved by using a large language model, and thus apply it to task planning. Unlike HSP, their approach focuses on completing the planning using the PDDL planner, and the LLMs are only used as an aid in generating the PDDL file, thereby underutilizing the LLMs' potential. Wang et al. [54] propose a Description, Explanation, Planning, and Selection (DEPS) framework for interaction planning based on a large language model. This framework helps to correct errors promptly during the long-term planning process and avoid unreasonable planning. This way complete reliance on LLMs for planning lacks a certain interpretability. Singh et al. [35] argue that task planning generated by a large language model can be made more plausible and feasible through a procedural prompts structure. We support this perspective and apply it to the HSP, making the planning program more logical and executable through code-based planning.

Although LLM-based task planning methods address certain limitations of classical planning, they inherently retain the drawbacks of LLMs [55, 56]. The design of the prompts is particularly important for such methods. In all of the above methods the respective designed cues are proposed, but the presence of factors such as the irrationality of the cue's design still leads the language model to make incorrect or infeasible planning for the task. Consequently, an innovative approach emerges through integrating classical planning methods with LLM-based planning, enabling agents to achieve planning with high and consistent success rates and strong generalization.

2.3. Integration of LLMs and PDDL Planning

A significant and relevant branch of research focuses on combining LLMs with PDDL-based planning to leverage the strengths of both paradigms. Existing approaches in this space can be broadly categorized by the role assigned to the LLM and the stage of PDDL utilization.

One prominent approach employs LLMs to generate the PDDL domain and problem files directly from natural language task descriptions, effectively using LLMs as a translator or modeler. For instance, Guan et al. [32] and similar works demonstrate that LLMs can produce syntactically correct PDDL constructs. However, the correctness and optimality of the generated PDDL are not guaranteed, as they rely entirely on the LLM's unverifiable internal knowledge and reasoning. Errors in the generated PDDL (e.g., incorrect preconditions/effects) propagate directly to the planning stage, leading to failure. In contrast, our HSP framework does not task the LLM with generating formal

PDDL files. Instead, we use a classical PDDL planner to produce a preliminary heuristic scheme and avoiding the inherent unreliability of LLM-generated PDDL.

Another approach, exemplified by [53], uses LLMs in a more limited capacity within a PDDL pipeline. In their work, the LLM is primarily used as an aid to parse natural language or to assist in generating parts of the problem specification, while the complete plan is derived by a traditional PDDL planner. The LLM's intervention is typically a one-time preprocessing step. The fundamental distinction between this method and HSP lies in the mechanism and phase of LLM intervention. While their approach confines the LLM to the initial file preparation phase, HSP integrates the LLM's generative capabilities deeply and recurrently into the planning process. Specifically, in HSP, the LLM operates in the Supplementary module to expand the stable heuristic plan into executable natural language steps, and in the Plan-grounding module to dynamically correct infeasible actions during execution. This creates a hybrid, interactive loop where classical planning provides a reliable scaffold, and the LLM dynamically adapts and refines it, which is a more extensive and integral utilization of the LLM's potential.

Consequently, HSP represents a distinct point in this methodological spectrum. It is neither a pure LLM-to-PDDL translation nor a PDDL-first method with light LLM pre-processing. Instead, it is a synergistic framework where a classically derived heuristic plan guarantees basic stability and correctness, and the LLM's commonsense knowledge and generative power are harnessed for plan supplementation and grounding. This innovative integration aims to achieve both stable and generalizable planning.

2.4. Do LLMs have planning capabilities?

Despite recent advances, the extent to which LLMs possess genuine, inherent planning abilities remains debated [57, 58]. Scholars find numerous unexplained challenges when using LLMs for planning, including hallucination [59], recency bias [60], and difficulties in calibrated uncertainty expression [61].

This skepticism has led many researchers to advocate for using LLMs not as standalone planners, but as auxiliary tools or knowledge sources within a larger system [20]. Frameworks like DoraemonGPT [62] use LLMs within a Monte Carlo Tree Search, while others employ LLMs as world models [63] or external commonsense knowledge bases [34] for agents.

The perspective that LLMs are better suited as powerful supplements rather than primary planners aligns with the design philosophy of HSP. Existing studies provide little evidence for robust, inherent LLM planning capabilities. Given their limitations, our framework deliberately avoids relying on the LLM as the core planner. Instead, HSP fully exploits the advantages of LLMs as supplements and modifiers. By using the LLM to complement the heuristic plan with commonsense details and to perform error correction during action execution, we aim to mitigate the planning instability caused by LLM hallucinations and other inherent shortcomings. Thus, HSP operationalizes the prevailing scholarly caution into a concrete architectural principle, utilizing LLMs in a manner that amplifies their strengths while containing their weaknesses.

3. Problem Formulation

Input: Our framework requires defining the target task T, the executable action space A for the agent, feedback image F, the initial state S_{init}, and the goal state S_{final}. Additionally, we provide task completion examples I in the context to improve the LLM's performance.

Output: Our goal is to use LLM to generate a complete sub-task sequence L through the input content. The task sequence is able to be directly grounded and executed in the Virtual Home [33] environment.

Example: For the task (T): *Refrigerate the Salmon*. We specify the set of actions that the agent is capable of performing A: *turnright, turnleft, walkforward, walktowards, walk, run, grab, switchon, switchoff, open, close, lookat, sit, standup, find, turnto, drink, pointat, watch, putin, putback*, the initial state S_{init}: *(at salmon kitchen), (at fridge kitchen)*, and the final state S_{final}: *(in salmon fridge)*. The given task example I is: *put the wine glass in the kitchen cabinet*. In addition, before executing the action *'Walk (kitchen)'*, HSP performs an assertion check and acquires a feedback image F (as shown in Figure 3). The assertion *'assert('Close to fridge')'* is evaluated, and the image-based feedback returns 'False', indicating the agent fails to meet the precondition.

Based on the above context, the final output sub-task sequences L of our framework for task T are *Walk (kitchen), Find (salmon), Grab (salmon), Find (fridge), Open (fridge), Putin (salmon, fridge), Close (fridge)*.

Table 1 provides a comprehensive summary of the main mathematical notations employed throughout this work, along with their formal definitions.

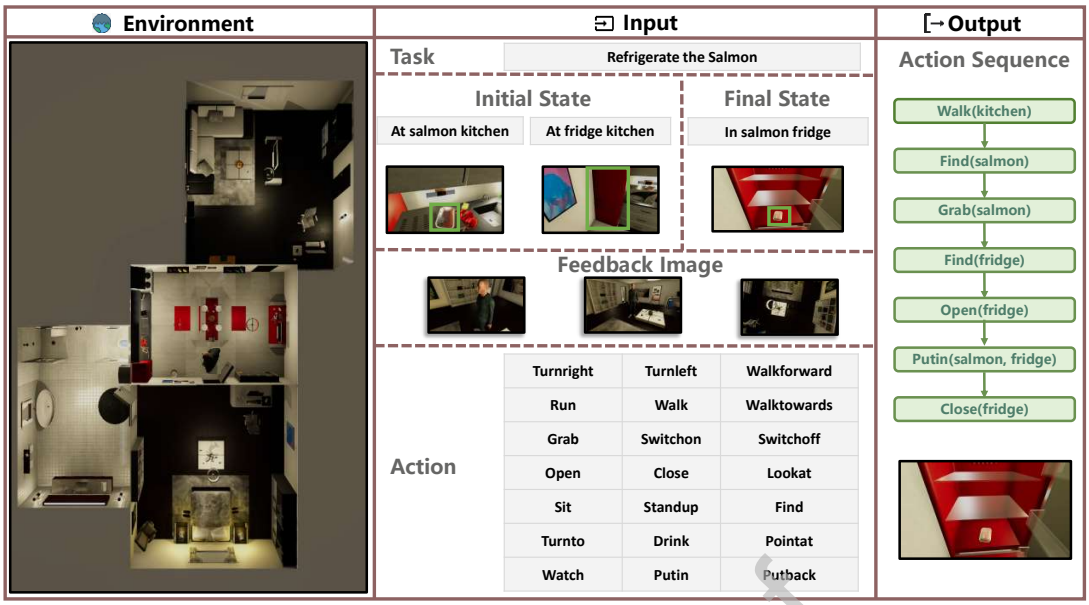

Figure 3: An example for input and output. The input and output structure of our framework is illustrated above. In this process, the initial and final states are used to derive heuristic planning, which guides the LLMs to generate the final executable action sequence.

Table 1
Key notations and descriptions used in this paper.

Notations	Definitions
T	Target task
A	Action space
F	Feedback image
S_{init}	Initial state
S_{final}	Final state
I	Task completion examples
L	Sub-task sequence

4. Methods

4.1. Overview

This section centers on the HSP framework. By leveraging the HSP framework, complex goal tasks can be decomposed into simpler, agent-understandable action steps that can be executed within a specific context. Within this framework, the `Heuristic` module (Section 4.2) is employed to guide the LLMs in generating planning outputs related to the goal tasks by providing heuristic suggestions. To enhance the plan generated by the `Heuristic` module, we introduce the `Supplementary` module (Section 4.3), which complements the initial heuristic plan by incorporating LLM-based augmentations. Finally, to facilitate the conversion of planning into agent-executable steps, we propose the `Plan-grounding` module (Section 4.4). Figure 4 illustrates the HSP framework.

While the high-level notion of decomposing complex tasks may appear superficially similar to the Chain-of-Thought (CoT) paradigm, HSP fundamentally differs in both architecture and mechanism. CoT operates as a sequential reasoning process internal to a single large language model, generating a linear chain of textual reasoning steps. In contrast, HSP is a modular, multi-stage planning-execution framework comprising three specialized components: (1) a formally verifiable PDDL planner (the Heuristic module) that guarantees logical feasibility; (2) a structured code-generation module (the Supplementary module) that refines and elaborates the plans produced by the Heuristic module; and (3) a grounding module (the Plan-grounding module) that performs runtime state verification and historical error correction. Therefore, HSP constitutes a more comprehensive planning framework.

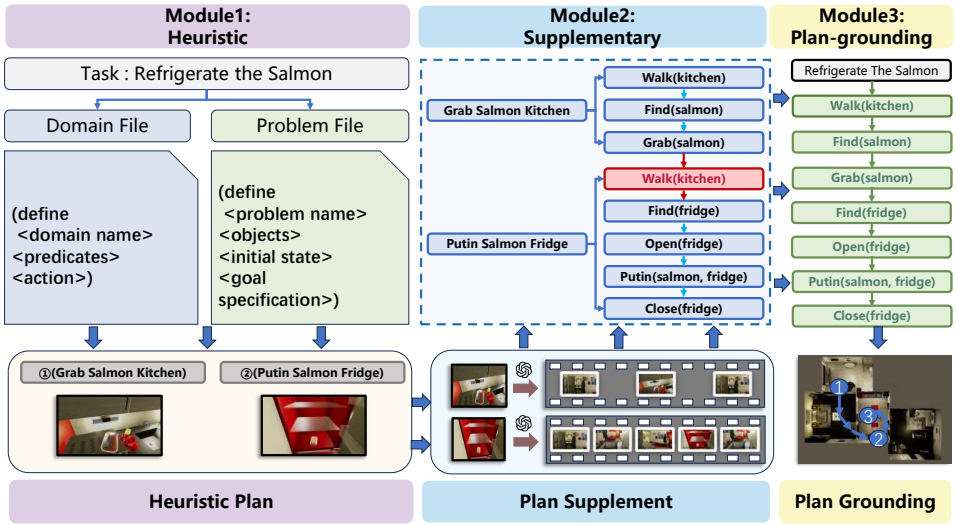

Figure 4: The framework of HSP. Our framework consists of three main parts: the `Heuristic` module, the `Supplementary` module, and the `Plan-grounding` module. The `Heuristic` module uses the PDDL planner to generate heuristic ideas, providing the initial task direction. The `Supplementary` module then enriches these heuristic ideas by leveraging LLM's generative capabilities. Finally, the `Plan-grounding` module is used to correct errors during execution and to complete the transition between task planning and the agent's actual actions.

4.2. Heuristic Module

When a large number of objects are present in the environment, direct task planning becomes challenging for LLMs due to the difficulty in establishing relationships between objects and tasks. Therefore, guiding LLMs through heuristic planning offers a viable solution. Heuristic planning does not require fine-grained task division; instead, it aims to focus the LLM's attention on the tasks themselves. Classical planning possesses the ability to exhaustively explore all feasible solutions, thereby producing an initial plan draft. Drawing inspiration from classical planning methods, our framework employs PDDL to generate an initial heuristic plan.

Planning with PDDL is a widely used approach for solving complex tasks[64, 65]. The approach explores feasible solutions in domain and problem files to find one or more valid plans. Nevertheless, crafting PDDL files for specific tasks demands extensive domain knowledge, as the number of actions and predicates to be defined varies depending on the task. Therefore, using PDDL to complete complex task planning inevitably requires substantial human involvement.

Excitingly, Guan et al. [32] explore the potential of using LLMs to assist in the generation of PDDL files. In the `Heuristic` module, we guide the LLM by prompting it to generate the necessary domain and problem files. The generated domain files must include the following components: a) predicates representing relationships. The predicates in the domain file define the attributes of objects. For instance, regarding the predicate *'is_open'*, after performing the *'open'* operation, the object *'fridge'* acquires the attribute *'is_open'*; b) actions that the agent is able to perform in the environment. After the agent performs these actions, it is necessary to describe whether the state of certain objects changes, which is typically done in conjunction with predicates; and c) a general classification of the items contained in the environment (*e.g., with container properties, room properties, etc.*). Used in generalized definitions. For example, both *'fridge'* and *'box'* possess the *'Container'* attribute and can thus be classified under the unified *'Container'* category. When defining predicates, only items in the *'Container'* category can be linked to the predicate *'is_open'*. An example of an LLM-generated domain file is shown in Figure 5.

We use LLMs to assist in generating the required problem files. The generated problem files need to include the following components: a) information about all the items contained in the environment. The problem file needs to specify all objects in the environment, implying that the environment is fully observable to the agent. PDDL generates an initial plan using the known environmental information; b) information about the initial state of the task items. The initial state description should only include task-relevant objects, with attention focused on their states. For example, for the task *'Put apple on the desk'*, only the initial states of the *'apple'* and *'desk'* need to be considered; and c) information about the final state of the task items. Corresponding to the initial state information, the state of the items associated with the task at the end needs to be given. An example of the prompt to generate a problem file is shown in Figure 6.

Instruct for the PDDL Domain File:

User: You are the professor in PDDL field. You need to complete the task: *Refrigerate the Salmon*, and You can only use these action: turnright, turnleft, walkforward, walktowards, walk, run, grab, switchon, switchoff, open, close, lookat, sit, standup, find, turnto, drink, pointat, watch, putin, putback.

User: Please complete the writing of the domain file.

The LLM:

```
(define (domain <domain name>)
        <PDDL code for predicates>
        <PDDL code for first action>
        [...]
        <PDDL code for last action>
)
```

Figure 5: An example of the prompt to generate a domain file.

Instruct for the PDDL Problem File:

User: You are the professor in PDDL field. You need to complete the task: *Refrigerate the Salmon*.

User: You can see 'plate', 'bananas', 'kitchentable', 'plum', 'window', 'lime' , ...in the environment.

User: The above items can be classified into three types: Object, Room, and Container.

User: Please complete the writing of the problem file.

The LLM:

```
(define (problem <problem name>)
        (:domain <domain name>)
        <PDDL code for objects>
        <PDDL code for initial state>
        <PDDL code for goal specification>
)
```

Figure 6: An example of the prompt to generate a problem file.

By using the above prompts, we obtain preliminary domain and problem files. However, it is important to note that LLMs' capability to generate PDDL files remains limited, particularly for smaller models such as GPT-3.5 [32]. Thus, manual refinement is required to improve the completeness and accuracy of the generated files. Although the `Heuristic` module still requires human intervention, LLMs can be used to construct an initial PDDL framework. Compared to manually crafting PDDL files, planning based on this framework exhibits enhanced stability.

We adhere to three explicit criteria for the manual optimization of PDDL domain and problem files: (1) Executability guarantee: ensuring that all action sequences are free of state conflicts and their preconditions are satisfiable; (2) Minimality principle: The optimized plan length resulting from heuristic planning excludes any complete, concrete planning chains and serves solely as a high-level heuristic guide. For instance, for the task eat chips on the sofa, the plan might only specify the essential high-level steps (grab chips kitchen) and (sit agent sofa). Details such as identifying which room the sofa is in, how to actually obtain the chips, and whether navigation to the kitchen is currently required are left for subsequent LLM-based supplementary modules to resolve through contextual reasoning and state-aware decision-making; (3) Goal reachability: the planner must be able to verify the existence of a feasible path from the initial state to the goal state.

After completing the acquisition of the PDDL files, we proceed to identify the necessary heuristic planning based on the planner. For the choice of the planner, we use the Fast Downward planner [66] in the LAMA branch. The planner generates an initial solution, which is simplistic and non-executable for agents. Fortunately, while this cannot be adopted as a final solution, it is able to provide a viable direction for LLMs to solve tasks.

4.3. Supplementary Module

It is inefficient for LLMs to understand relevant task descriptions and environment states through natural language. As a result, the planning solutions they generate are often difficult for agents to execute, which reduces the agents' task success rate [36]. Singh et al. [35] demonstrate that the executable nature of LLM-generated plans can be improved through the use of encoding structures. In the `Supplementary` module, we utilize code structure hints to enable the LLM to generate task planning solutions, and it is intended to complement the planning solutions generated by the PDDL planner. In this section, the LLM generates more detailed and enriched plans based on the heuristic outputs from the `Heuristic` module. The generated planning functions include calls to action library, comments summarizing the actions, and assertions. An example of the planning generated by the LLM is shown in Figure 7.

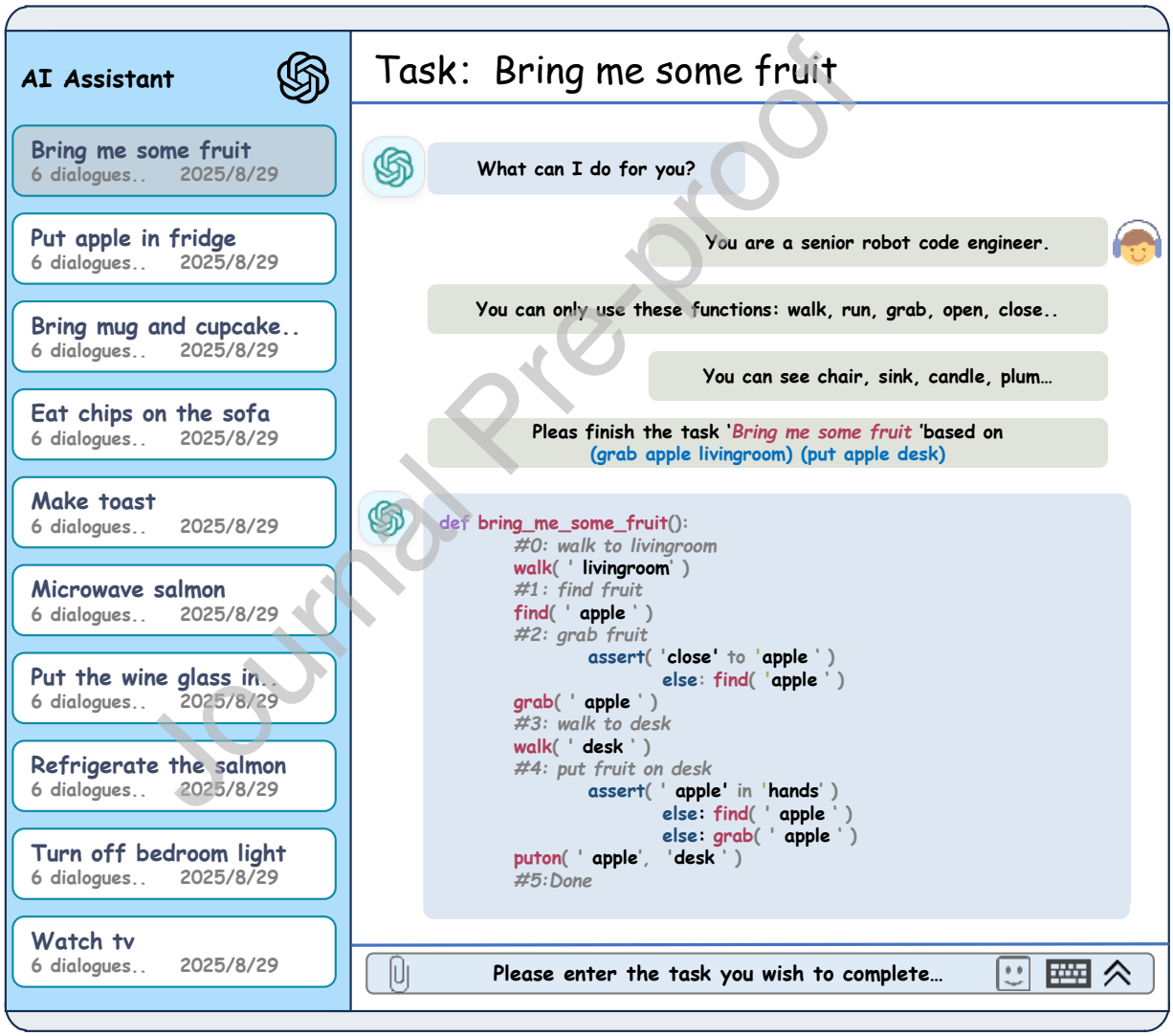

Figure 7: The Code-structured plan generation process. During interaction, the agent instructs the LLM to function as a code engineer. It is expected that the LLMs will tell the agent how to complete the target task in a code structure. Through prompts, the agent specifies its executable actions and interactable objects. The LLM subsequently generates the final task plan by refining the heuristic planning.

First, LLMs refine the plan by utilizing the action library based on the initial options provided by the *Heuristic* module. LLMs state the two primary components when generating calls to the action library. The first component

involves the actions the agent can perform, such as *'walk'* or *'find'*. The second component specifies the items or contexts required to execute these actions. For instance, the *'walk'* action is associated with a location, such as *'livingroom'*, while the *'putin'* action depends on items like *'apple'* and contexts like *'desk'*. Additionally, annotating actions improves the interpretability of LLMs' outputs. By explaining each planning step alongside its execution, LLMs partially enhance the logical coherence and rationality of their solutions [35].

The plans generated by LLMs may contain some illegal operations, depending on the definition of the environment. For example, for the *'sit'* operation, it is sufficient to simply use *'sit'* rather than specifying an object, e.g., *'sit sofa'*, and the agent will automatically find an item to sit on based on the surrounding items. Nevertheless, LLMs may plan some illegal actions due to limited knowledge of the environment. To distinguish between executable and illegal actions, we classify the actions into three categories based on the number of operational items involved: no action items, one action item, and two action items. For actions in the no-operation-item category, they are actions that are used to change the agent's state, e.g. *'sit'*, *'watch'*, etc. Actions that operate on one item, on the other hand, require the agent to interact with such an item, e.g., *'find apple'*, *'grab apple'*, etc. Actions involving two items typically involve an interaction between the items through the agent, as in *'putin apple fridge'*, *'putback apple desk'*, etc. These three categories encompass all executable actions for the agent. Any planned action that does not correspond to one of these categories is considered an illegal operation. When planning output by the LLMs includes illegal actions, the agent will match them to the action library and correct the illegal actions.

Due to the gap between planning and execution, not all operations are executed in the environment; it is therefore important to ensure that critical steps are not overlooked during execution. We add an assertion mechanism to the *Supplementary* module, enabling agents to verify conditions during critical steps. When an assertion is triggered, the current state is submitted to the LLM as a new contextual cue, enabling it to determine whether the previous step should be repeated. The assertion mechanism improves the agent's task completion success rate. By enabling the agent to determine whether the current state satisfies the task conditions, the assertion mechanism allows the agent to revisit its state using the LLM to decide if additional steps are required to obtain critical items.

Algorithm 1 Supplementary Module Refinement Process

Require: High-level PDDL plan from HEURISTIC module
Ensure: Executable code-structured plan

procedure REFINEPLAN(P_{high})
  for each action $a_i \in P_{\text{high}}$ **do**
    Decompose a_i into low-level primitives
    Insert precondition assertions
    Add failure recovery routines
  end for
  return $P_{\text{executable}}$
end procedure

Example: Task: "Put an apple on the desk"
P_{high}: `(grab apple livingroom)` → `(put apple desk)`
$P_{\text{executable}}$:
```
walk('livingroom')
find('apple')
assert('close' to 'apple')
if assertion fails: find('apple')
grab('apple')
... (similarly for put action)
```

To intuitively illustrate the refinement process of the `Supplementary` module, we provide a concrete example based on the task "Put an apple on the desk" (as shown in Algorithm 1).

Initially, the `Heuristic` module generates a high-level PDDL-style plan: `(grab apple livingroom)` followed by `(put apple desk)`. While logically sound, this plan is non-executable for a physical agent because it lacks transitional actions (e.g., navigating to the object) and fail-safe mechanisms.

The `Supplementary` module transforms this skeleton into a code-structured plan. For instance, the high-level step `(grab apple)` is expanded into a sequence: `walk('livingroom')`, `find('apple')`, and `grab('apple')`. Crucially, an assertion `assert('close' to 'apple')` is inserted before the grab action. If the agent is not close enough due to environmental dynamics, the module triggers a recovery behavior (`else: find('apple')`). This refinement ensures that the abstract heuristic guidance is grounded into precise, context-aware, and robust executable commands.

The core of the Supplementary Module is a rule-based adaptive parameter parser, whose architecture is structured around the following design guidelines: (1) Input Parsing: it receives the tokenized action sequence output by the LLM, identifying action predicates and noun phrases; (2) Parameter Parsing and Binding: guided by an action taxonomy (e.g., navigation actions require one location parameter, placement actions require two to three parameters), it performs type-driven object search within the environmental scene graph, mapping textual nouns to concrete object IDs; (3) State Verification and Implicit Supplementation: before execution, it checks the state constraints of each action (e.g., verifying the holding state before executing putin). If constraints are unsatisfied, necessary preparatory actions (e.g., grab) are automatically inserted at the head of the plan; (4) Standardized Output Generation: finally, all actions are formatted into a unified executable script according to the 'VirtualHome Command Mapping Specification', ensuring direct interpretability by the simulation environment.

4.4. Plan-grounding Module

The above guidance allows LLMs to generate planning in a more logical manner. However, it is important to note that the planning produced by the LLMs is not directly understood by the agents in the environment. Therefore, a conversion process is required to translate the planning into executable actions for the agents. This conversion ensures that the generated plans are executable within the environment. We propose `Plan-grounding` module to implement this process. Before each action is executed by the agent, the `Plan-grounding` module analyzes the feasibility of the action, corrects errors, and removes redundant operations.

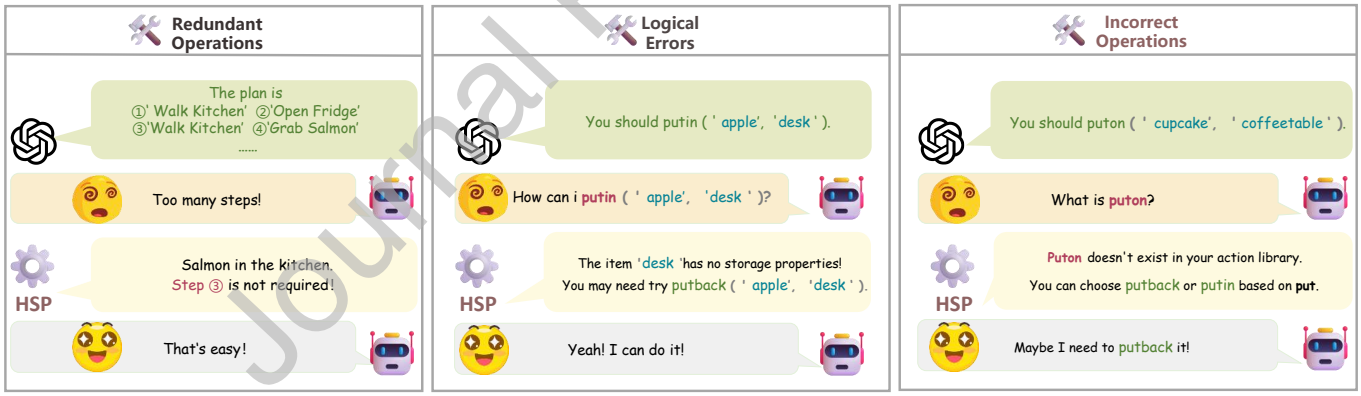

Figure 8: Actions in the HSP. Actions that are supplemented by the LLM may not always be fully executed. Our framework identifies and corrects actions that fail during `Plan-grounding` section. These errors occur primarily due to mismatches with item properties or because the actions are absent from the agent's library.

Concretely, we find that LLMs' understanding of the state of the environment is not so satisfactory. This results in action sequences that are not always executable by the agent. We systematically summarize these problems and propose targeted solutions. In the `Plan-grounding` module, the following three types of problems are targeted: **a) Redundant operations**; **b) Logical errors**; **c) Incorrect operations**. Figure 8 shows an example of our use of `Plan-grounding` to help agents accomplish tasks.

Adding conditional judgments to resolve redundant operations caused by LLMs hallucination problem. The success rate of long-term task execution tends to decrease as the number of execution steps increases. To improve the agent's success rate in task completion, it is essential to eliminate redundant operations during the execution process.

In our framework, redundant operations primarily arise from LLM hallucinations [67]. Even though we give LLMs sufficiently well-developed prompts to remind them of the current state and the actions that have been completed, LLMs will still repeat the same actions and cause the task to fail. Ye et al. [68] demonstrate that it is unrealistic to try to completely eliminate the hallucination problem in LLMs. Fortunately, the hallucination problem can be mitigated through conditional judgments. To systematically identify and skip redundant actions, the Plan-grounding module implements a threshold-based filtering algorithm. It monitors the execution history H to detect two primary types of redundancy: (1) consecutive repetitions of the same action-object pair (e.g., two sequential `walk(kitchen)` commands), and (2) excessive executions of the same action type toward a semantically identical target (e.g., navigating to the same room multiple times during a single task episode). Based on the characteristics of the task domain, repeated navigation to the same room is typically erroneous; therefore, a redundancy threshold θ_a is defined empirically for each action type a. For navigation, the threshold is set to $\theta_{\text{walk}} = 1$, reflecting the observation that successful tasks seldom require the agent to walk to the same room more than once unless a distinct subgoal (e.g., retrieving a different object) justifies it. This mechanism effectively terminates hallucination-induced loops. Specifically, in the `Plan-grounding` module, our framework keeps track of the currently executed actions, and if a certain number of times is exceeded, the action is skipped and the next action is executed. Take the task *'Refrigerated salmon'* executed by the agent as an example, if some of the task steps are *'Walk Kitchen', 'Grab salmon ', 'Walk Kitchen', 'Put salmon'*, then the second *'Walk Kitchen'* is a redundant step for our framework and needs to be removed. This is because the *'salmon'* exists in the *'kitchen'* and does not need to be brought back into the kitchen.

The matching mechanism assists the agent analyze item properties to resolve logical errors. Logical errors can occur during task execution, primarily due to the agent's inability to accurately interpret the current state information. In the Virtual Home environment [33], items of the same type are differentiated by unique numerical identifiers. This introduces the possibility that the agent may acquire items that are inconsistent with the intended ones, leading to task failure. For example, in the task *'Put apple on the desk'*, there are numerous apples in the environment, and their distinction relies on their respective numbers. Even though both items are apples, *'apple 437'* is not the same as *'apple 438'*. The agent may attempt to perform the task with operations such as *'Grab Apple 437'* and *'Put Apple 438'*, which result in task failure. To address this issue, we employ the `Plan-grounding` module to assist the agent in keeping track of item numbers. Specifically, we record the item number when the agent performs the *'get'* action and ensure it matches when the agent performs the *'put'* action. If the match is inconsistent, the number of the item obtained takes precedence. Additionally, a logical error arises from the LLM's inability to comprehend the properties of items in the environment. We find that LLMs cannot distinguish between *'putin'* and *'putback'* operations due to their failure to correctly interpret item attributes. For items with container attributes, the agent can perform *'putin'* operations, while for items without container attributes, only *'putback'* operations are appropriate. We resolve this issue by evaluating the item attributes and selecting the appropriate actions based on these attributes, thereby correcting logical errors in the planning process.

Utilizing keyword extraction to complete correction of incorrect operations. When LLMs generate plans containing actions outside the agent's action library, they introduce incorrect operations. Therefore, we use the extraction of keywords in error action correction. We classify the ambiguous *'put'* type of actions and add judgment conditions based on the attributes of the items, so as to correct these erroneous actions. For actions that neither exist in the agent's action library nor can be categorized by us based on the keywords, we correct this during execution through the assertion mechanism.

In addition, we find that visual information is more effective than using textual information to determine assertion checks. By utilizing visual information, agents prioritize processing target objects that are highly relevant to the current task. This information processing method not only effectively reduces the high complexity problem caused by text description during the feedback process, but also significantly improves the accuracy and execution efficiency of task decision-making. A complete assertion checking process is shown in Figure 9. During assertion checks, the agent inputs the environmental status in the form of images into the VLMs. Then, VLMs analyze the content in the image based on the current task and assertion checks to determine whether the agent complete the current task node. If it is determined based on the image content and assertion checks that the current task node is completed, VLM outputs 'True' and skips the fix action. Otherwise, it outputs 'False' and corrects the current state.

Overall, the Plan-grounding Module implements a three-layer progressive repair framework for runtime error detection and correction. In the first stage, if a grab(obj) action fails due to object ID mismatch or inaccessibility, the system recovers the correct object identifier by backtracking through the execution history, based on the Object History Tracking Hypothesis. The second stage employs a threshold-based redundancy filtering mechanism that automatically

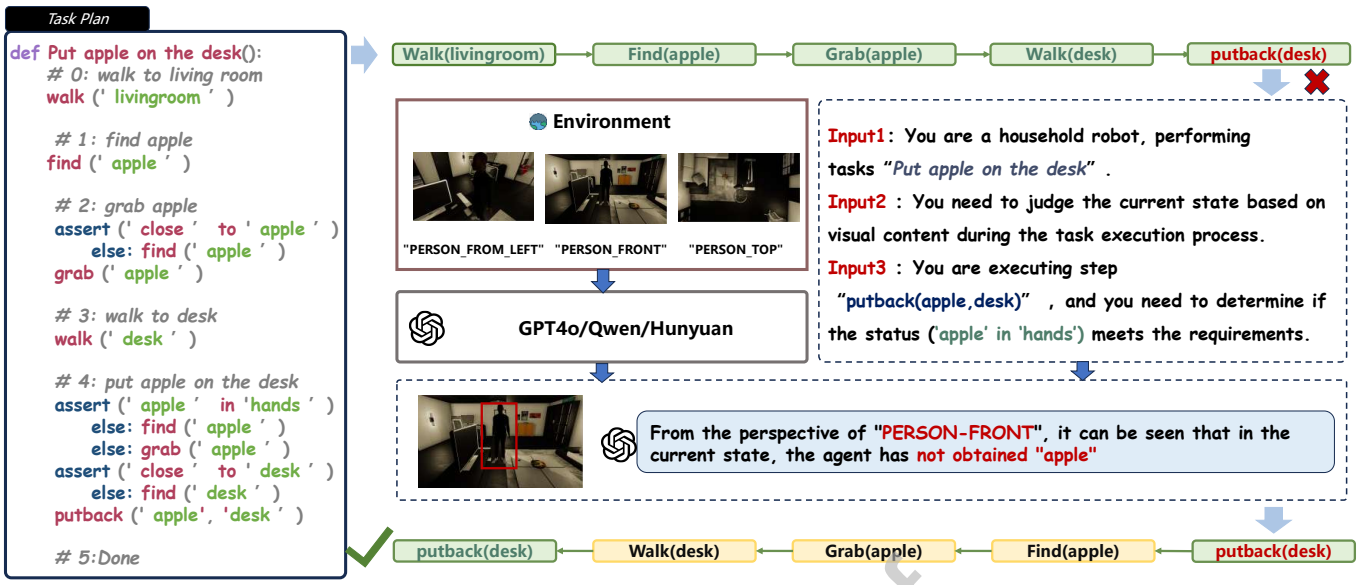

Figure 9: Action correction. Action correction verifies the current state to prevent critical task failures. When enabled, it aborts the task sequence, avoiding further deviation from the expected state due to erroneous executions.

identifies and eliminates repeated navigation actions to the same location—for instance, retaining only the first walk command to a given room. In the third stage, each action undergoes pre-execution scene-graph validation to confirm the actual existence of referenced objects in the environment; actions referring to non-existent objects are skipped and logged. All repair operations are embedded within a continuous state-update loop: after each successful action execution, the system synchronously updates the agent's internal state (e.g., location, inventory), thereby providing accurate contextual state information for subsequent constraint verification. This ensures the ongoing feasibility and consistency of the plan in dynamic environments.

4.5. Addressing Environmental Artifacts through Principled Task Design

Our evaluation is centered on clearly assessing the core planning capabilities of HSP. We acknowledge that the VirtualHome environment, like any complex simulator, contains known bugs and inconsistencies (e.g., the documented failure of the `open(washingmachine)` action in Environment 0). To ensure that the results reflect planning performance rather than compatibility with specific simulator imperfections, we adopt a **principled task selection strategy**. HSP is evaluated on a curated set of challenging tasks in which all required object interactions are verified to be functionally sound and reproducible within the simulator. This approach effectively isolates HSP's planning intelligence from confounding environmental artifacts. The framework's ability to successfully complete tasks within this verified subset, without interference from environmental factors, robustly demonstrates its superior planning capability.

5. Experiments

5.1. Experiment Setup

Experimental environment. For our experimental environment, we chose Virtual Home, a simulation environment that features a wide variety of daily life scenarios and objects. In this simulation environment, the agent needs to understand the task requirements and select the most task-relevant objects from a vast array of options and complete the task objectives. On the selection of target tasks, we refer to the planning task in Ming et al.'s [36] study. Although the study takes enough thought to the selection of tasks, there are still some tasks that, though seemingly achievable, are infeasible due to Virtual Home's limitations. As a consequence, we reformulate 10 tasks for agent. Among these ten tasks, there are short-step tasks like *Turn off the bedroom light* which requires only 3 steps. There are also long-step tasks like *Microwave salmon* which require 8 steps.

Baselines: To evaluate whether our framework achieves better performance in task planning, we compare it with the following baseline frameworks.

- **ProgPrompt [35]:** The baseline ProgPrompt adopts a distinct response format. Unlike other prompts that directly guide LLM outputs, ProgPrompt posits that code-structured responses enhance LLMs' reasoning and task completion. To enhance LLM effectiveness in task planning, we provide two exemplars of successful problem-solving.
- **HiCRISP [36]:** The baseline HiCRISP has the same output format as ProgPrompt, but introduces a feedback mechanism during execution. To implement the feedback link, HiCRISP leverages a stack-based approach. By employing the LLM to analyze the current state and resolve issues, it improves task completion success rates. We include two problem-solving examples to guide the LLM.

Metrics: For the evaluation metrics, we adopt metrics from prior studies [36, 35] and make improvements based on real-world applicability. Meanwhile, in order to verify whether our framework is able to successfully guide LLMs to think and perform their tasks better by considering the logic of executing tasks, we design other additional evaluation metrics. The specific metrics are shown below:

- **Success Rate (SR):** In order to assess whether the agent is able to accurately accomplish the target task, we introduce SR. Before completing the task, the agent must satisfy all associated task conditions, and SR is the execution score that assesses whether the agent achieves all the goal conditions associated with the task.

$$SR = \frac{Num_{HF}}{Num_{AF}}, \tag{1}$$

  where Num_{HF} represents the number of goal conditions the agent has already completed and Num_{AF} represents the total number of goal conditions the agent needs to complete for the given task.

- **Executability ($Exec$):** Since the target task planning is generated by the LLM, we introduce the metric $Exec$ to evaluate whether the generated actions are aligned with the environment. A higher $Exec$ indicates a greater likelihood of successful execution of LLM-generated plans by the agent, even if the actions are not task-related.

$$Exec = \frac{Num_{SE}}{Num_{AE}}, \tag{2}$$

  where Num_{SE} represents the number of actions successfully executed by the agent within the environment, and Num_{AE} denotes the total number of actions planned by the framework for this task.

- **Average Execution Steps ($|A|$):** An efficient agent should accomplish a task with minimal steps. To evaluate this, we propose the metric $|A|$, defined as:

$$|A| = \frac{S_t}{N_t}, \tag{3}$$

  where S_t represents the sum of steps for each task and N_t represents the number of times the task is performed.

To evaluate the performance of different Visual Language Models(VLMs) in household task scenarios, we propose the following evaluation metrics.

- **Accuracy (Acc):** We evaluate VLMs' veracity judgment capability via Acc.

$$Acc = \frac{Num_{\text{correct}}}{Num_{\text{total}}}, \tag{4}$$

  where N_{correct} is the number of correctly classified assertions, and N_{total} the overall evaluation count.

Other experimental details: All our experiments are conducted using GPT-4 [69]. LLMs are not fine-tuned for specific tasks in our experiments. The version of the simulation environment Virtual Home [33] we use is 2.3.0 and we build it on Ubuntu 20.04. In our experiment, a total of 10 tasks are designed. These tasks include *Bring mug and cupcake to the coffee table, Eat chips on the sofa, Make toast, Microwave salmon, Put apple in fridge, Put apple on desk, Put the wine glass in the kitchen cabinet, Refrigerate the salmon, Turn off bedroom light, Watch TV*. Note that LLMs is more context-sensitive, and to ensure the rigour of our experiments, we try to ensure consistency of the prompt examples in our experiments. As shown in Table 2.

Table 2
Introduction to the tasks.

Tasks	Descriptions	True Steps
Task1	Bring mug and cupcake to the coffee table	8.0
Task2	Eat chips on the sofa	5.0
Task3	Make toast	6.0
Task4	Microwave salmon	8.0
Task5	Put apple in fridge	8.0
Task6	Put apple on desk	5.0
Task7	Put the wine glass in the kitchen cabinet	7.0
Task8	Refrigerate the salmon	7.0
Task9	Turn off bedroom light	3.0
Task10	Watch TV	5.0

5.2. Experimental Results

In this section, we analyze the effectiveness of different VLMs in judging assertion states and perform detailed experimental comparisons. Specifically, we evaluate the VLMs using feedback images from 10 tasks to test the VLMs' assertion-checking performance. The experimental results are shown in Table 3. Additionally, we compare in detail the performance of our task planning framework (**HSP**) with the baseline frameworks based on Tables 4 to 5. Experimental results demonstrate that HSP outperforms baselines, achieving both a **higher success rate** and **fewer required steps**.

Comparison of VLMs Performance: We extend our evaluation to five visual language models (VLMs): GPT-4o [69], Qwen2.5-vl-72b-instruct [70], Hunyuan-vision [71], ernie-4.5-turbo-vl-32k [72], and llama-3.2-90b-vision-instruct [73]. As shown in Table 3, GPT-4o demonstrates the highest overall accuracy among open-source models (80%), substantially outperforming Qwen2.5-vl-72b-instruct (25.5%). Among all the models, GPT-4o achieves the highest accuracy overall, with the task "put the wine glass in the kitchen cabinet" reaching 100% accuracy, while "put apple in fridge" performs poorly at only 66.7% due to misidentification of the fridge in the virtual environment. These results indicate that model size, vision-language alignment, and task-specific recognition capabilities strongly influence performance, highlighting the importance of careful model selection when using VLMs as visual feedback modules. Compared to Qwen2.5-vl-72b-instruct and Hunyuan-vision, GPT-4o consistently yields superior results in assertion-checking tasks.

Nevertheless, both llama-3.2-90b-vision-instruct and ernie-4.5-turbo-vl-32k exhibit substantial limitations relative to GPT-4o's benchmark performance of 80% accuracy. llama-3.2-90b-vision-instruct performs relatively well in multi-object manipulation tasks, such as "bring mug and cupcake to the coffee table" and "refrigerate the salmon" (both 75% accuracy), whereas ernie-4.5-turbo-vl-32k shows relative proficiency in simpler tasks like "watch TV" (100% accuracy) and "turn off bedroom light" (50% accuracy). Both models, however, struggle considerably with certain tasks: ernie-4.5-turbo-vl-32k completely fails "put the wine glass in the kitchen cabinet" (0% accuracy), while llama-3.2-90b-vision-instruct fails "make toast" and "put apple in fridge" (0% accuracy). These failures are likely due to challenges in object recognition, spatial reasoning, or state transition understanding within the virtual environment.

The observed variation in performance across different VLMs underscores the critical importance of model selection for visual feedback systems. Although llama-3.2-90b-vision-instruct shows promising results among open-source alternatives, its performance remains considerably lower than proprietary models such as GPT-4o. This suggests that current open-source VLMs still face limitations in complex visual reasoning tasks, particularly those requiring precise object localization, state tracking, and multi-step reasoning in virtual environments.

Table 3
Comparison of VLMs performance.

VLMs	Correct Number	Total Number	Accuracy
GPT-4o	44	55	0.800
Qwen2.5-vl-72b-instruct	14	55	0.255
Hunyuan-vision	21	55	0.382
ernie-4.5-turbo-vl-32k	16	55	0.291
llama-3.2-90b-vision-instruct	25	55	0.455

HSP vs. ProgPrompt: As shown in Table 4, the ProgPrompt framework modifies the output format of LLMs at the output level by adopting a code-based structured output approach. This method demonstrates certain improvements in performance. Regarding task completion, ProgPrompt successfully accomplishes all 10 designated tasks. Analysis of the three evaluation metrics reveals that while ProgPrompt generates planning solutions for all tasks, it cannot guarantee the executability of these plans. Particularly in Task 1, ProgPrompt exhibits significant instability. During multi-step planning processes, although LLMs provide relatively detailed planning solutions, the agent fails to execute them successfully in the environment, resulting in a SR of 0 across multiple task executions. Furthermore, ProgPrompt's excessive focus on assertion mechanism verification leads to continued attempts of already completed steps, even when the agent has acquired all necessary conditions for task completion. This behavior increases the number of execution steps and consequently elevates task difficulty. In contrast, the HSP framework benefits from its `Plan-grounding` module, which enables successful conversion of LLM-generated plans into executable agent actions and ensures timely termination upon task completion.

Table 4
Planning Performance of HSP and ProgPrompt in Virtual Home.

Tasks	ProgPrompts			HSP(Ours)		
	SR↑	Exec↑	|A|↓	SR↑	Exec↑	|A|↓
Task1	0.180	**0.819**	9.80	**1.000**	0.790	**8.40**
Task2	0.892	0.918	6.80	**0.977**	**0.967**	**6.00**
Task3	0.596	0.845	11.20	**0.983**	**0.908**	**9.20**
Task4	0.714	0.911	16.00	**0.913**	**1.000**	**8.00**
Task5	0.571	0.906	11.20	**0.941**	**0.971**	**8.80**
Task6	0.383	0.790	**5.60**	**1.000**	**0.800**	6.00
Task7	0.904	0.753	11.60	**0.976**	**0.756**	**8.00**
Task8	0.771	**1.000**	8.40	**0.959**	**1.000**	**7.00**
Task9	**0.968**	0.637	3.80	0.899	**0.667**	**3.40**
Task10	0.949	0.667	8.40	**0.954**	**0.790**	**6.00**

HSP vs. HiCRISP: Regarding HiCRISP (as presented in Table 5), the framework incorporates error consideration during execution and introduces a feedback loop, which partially compensates for deficiencies in LLMs planning. However, experimental results reveal that when employing GPT-4 as the LLM, the framework occasionally produces unsatisfactory planning outputs due to inherent limitations in LLM capabilities. These outputs cause the HiCRISP framework to inaccurately capture planning content, ultimately leading to task planning failures. A specific manifestation occurs in Task 6, where the HiCRISP framework fails to explicitly guide the agent that the apple is located in the living room. When relying solely on LLMs for planning, their commonsense knowledge defaults to placing apples in the kitchen, causing immediate task failure and consequently yielding low SR for HiCRISP. In contrast, the HSP framework benefits from its `Heuristic` module, which explicitly specifies the apple's initial location during heuristic planning, thereby achieving significantly higher success rates for Task 6. Analysis of average execution steps demonstrates that HiCRISP requires the highest number of steps across all tasks. This stems from corrective actions implemented within HiCRISP. While these corrections positively affect $Exec$, they may conversely reduce SR. The stack mechanism integrated into HiCRISP proves effective in facilitating successful execution of LLM-generated plans within the environment. Nevertheless, for planning problems specifically, HiCRISP underperforms compared to HSP. This comparative analysis suggests that adopting the HSP framework can enhance LLMs' task planning capabilities.

Findings: **The experimental results show that HSP performs better compared to the other baseline frameworks.** Figure 10 illustrates the performance comparison of the baseline frameworks with HSP. The assertion mechanism in ProgPrompt necessitates additional steps. By introducing a stack mechanism, HiCRISP performs well on various tasks and significantly improves the indicator of action execution success rate. For our HSP framework, it is superior to the baseline frameworks mentioned above. Our framework helps LLM better understand tasks through heuristic planning, thereby improving the success rate of task completion on most tasks. Furthermore, the synergistic

Table 5
Planning Performance of HSP and HiCRISP in Virtual Home.

Tasks	HiCRISP			HSP(Ours)		
	SR↑	Exec↑	|A|↓	SR↑	Exec↑	|A|↓
Task1	0.542	0.767	13.6	**1.000**	**0.790**	**8.4**
Task2	0.970	0.753	10.6	**0.977**	**0.967**	**6.0**
Task3	0.785	0.893	10.6	**0.983**	**0.908**	**9.2**
Task4	0.709	0.902	12.8	**0.913**	**1.000**	**8.0**
Task5	0.878	0.911	10.2	**0.941**	**0.971**	**8.8**
Task6	0.293	0.684	9.2	**1.000**	**0.800**	**6.0**
Task7	0.768	**0.989**	11.2	**0.976**	0.756	**8.0**
Task8	0.809	0.875	7.8	**0.959**	**1.000**	**7.0**
Task9	0.756	0.653	5.6	**0.899**	**0.667**	**3.4**
Task10	0.936	0.769	10.4	**0.954**	**0.790**	**6.0**

effect of the `Heuristic`, `Supplementary` and `Plan-grounding` modules enables agents to achieve goals with minimal steps.

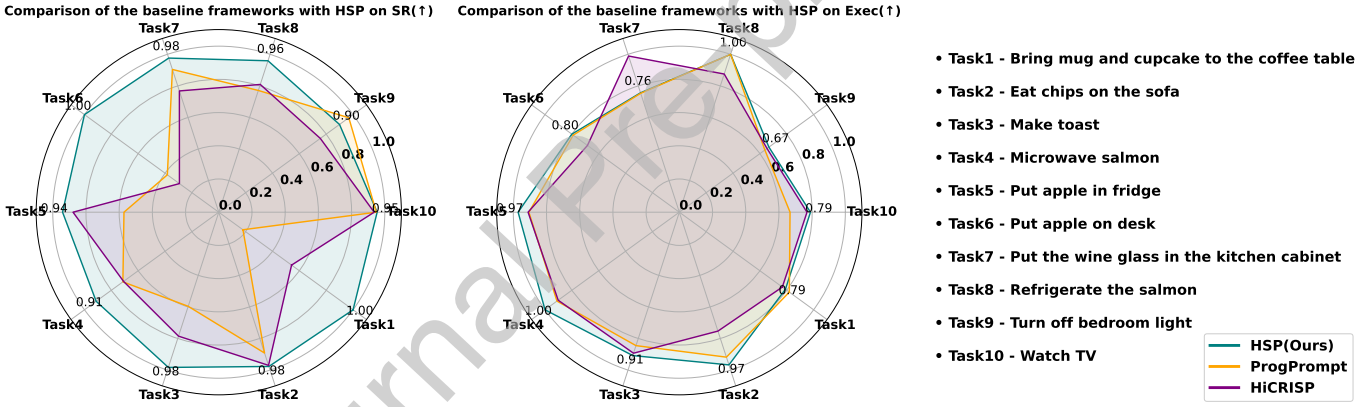

Figure 10: Comparison of the performance of the baseline frameworks with the HSP. For the framework, higher SR values correlate with greater task completion, while higher $Exec$ scores reflect better action acceptability.

Ablation Study: To assess the importance of each part of the framework HSP for task planning, we conduct ablation experiments. Table 6 and Figure 11 present our ablation experimental results.

Heuristic module assists agents in better understanding the relationship between tasks and the environment. From the experimental results, the removal of Heuristic module is detrimental to the planning of LLMs, which leads to a decrease in SR. In our framework, the main role of heuristic planning is to help LLM better understand the relationship between environment and task. In Table 6, it is found that HSP w/o Heuristic is almost difficult to accomplish for Task 1 and Task 2. This demonstrates that LLM alone cannot effectively discern the relationship between environmental context and task requirements from the prompt. However, incorporating heuristic planning enables proper discovery of these task-environment relationships.

Supplementary module is the bridge between heuristics and actual planning, it is also the basis of the HSP framework. Removing Supplementary module theoretically leads to a significant decrease in all metrics. Whereas, this does not occur in practice. Although the success rate of tasks when the Supplementary module is removed does not appear to be decreasing. This does not mean that the actions performed by the agent are reasonable. For example for the task *'Watch TV'*, the heuristic planning only gives the steps *'switchon TV'* and *'sit sofa'*. Nevertheless, it is unreasonable for the agent to perform only these two steps because it does not find the location of *'TV'* and *'sofa'*.

Table 6
Ablation Experiment of HSP.

Tasks	HSP w/o Heuristic			HSP w/o Supplementary			HSP w/o Plan-grounding		
	SR↑	Exec↑	|A|↓	SR↑	Exec↑	|A|↓	SR↑	Exec↑	|A|↓
Task1	0.154	0.656	10.00	**0.932**	0.500	**4.00**	0.417	**0.667**	9.00
Task2	0.192	**0.200**	5.00	**0.899**	0.000	2.00	0.000	0.000	**0.00**
Task3	0.974	0.916	8.20	0.988	0.800	**3.00**	**1.000**	**0.950**	6.40
Task4	0.745	**0.850**	18.40	**0.954**	0.840	5.00	0.000	0.000	**0.00**
Task5	0.265	**0.378**	9.00	**0.927**	0.250	4.00	0.000	0.000	**0.00**
Task6	0.274	**0.837**	4.80	0.937	0.000	**2.00**	**0.980**	0.800	5.00
Task7	0.868	0.667	9.00	0.741	0.500	**4.00**	**0.972**	**0.857**	7.00
Task8	0.874	0.803	13.20	0.903	0.850	**4.00**	**1.000**	**1.000**	7.00
Task9	0.954	**0.667**	3.80	0.025	0.500	**2.00**	**0.982**	**0.667**	3.00
Task10	0.937	**0.676**	6.20	0.887	0.200	**2.00**	**0.952**	0.653	6.00

The SR metric remains remains high because the agent completes most critical steps and achieves the majority of key node states specified in the task. Although the HSP w/o Supplementary allows the agent to complete the task in the least number of steps in tasks, while the fulfilment of these tasks is practically incorrect.

The Plan-grounding module assists in correcting misoperation and improves the efficiency of the agent in accomplishing tasks. The evaluation metric $Exec$ is closely tied to the execution of actions in the environment, and the primary purpose of the Plan-grounding module's design is to address the framework's execution issues in the environment. Therefore, removing the Plan-grounding module leads to a decline in the agent's execution accuracy and success rate. By analyzing the experimental results, we observe that HSP w/o Plan-grounding exhibits a slight improvement in executability for specific tasks, which stems from the agent's repeated execution of completed steps during task performance. As for the Plan-grounding module, its introduction assists the agent in successfully breaking out of loops and accomplishing tasks. This is particularly evident in task 5, where the agent is unable to complete the task after removing the Plan-grounding module.

Vertical Comparison and Task Diversity. As illustrated in Figure. 12(b), the tasks exhibit varying sensitivities to the modules. *Environment-dependent tasks* (e.g., T5) show a significant performance drop in the "w/o Plan-grounding" setting, as they involve dynamic interactions that require precise P-module alignment. In contrast, *Logic-intensive tasks* (e.g., T1 and T6) rely heavily on the H-module (Heuristic) to maintain a coherent sequence over long horizons. For simpler, *Goal-oriented tasks* (e.g., T3 and T8), the Success Rate (SR) remains relatively high across all ablation variants in Fig. 12(b), implying that the inherent reasoning of LLMs is sufficient only for low-complexity scenarios.

Statistical Insights on Metrics Correlation. The interaction between $Exec$ and $|A|$ in Fig. 12(a) highlights the stability of the HSP framework. In the "w/o Heuristic" setting, the red dashed line representing average $|A|$ exhibits high variance (ranging from 3.8 to 18.4), demonstrating that LLMs without PDDL guidance generate inconsistent and often redundant plans. Furthermore, the decoupling of SR and Exec is clearly visible in the "w/o Supplementary" results of Figure. 12(a)—where SR remains high despite a sharp decline in Exec. This gap uncovers a tendency for LLMs to ignore physical constraints, such as acting on an object without first locating it.

Stability and Planning Redundancy. Data from Table 6 and Fig. 13(a) reveal that without the PDDL-based heuristic module (*w/o Heuristic*), the framework suffers from significant planning instability. The plan length $|A|$ fluctuates drastically from 3.8 to 18.40 steps. This high variance, particularly in Task 4, suggests that the H-module is essential for constraining the LLM's stochastic output and eliminating redundant reasoning loops, thereby ensuring a deterministic and optimal plan trajectory.

Efficiency-Success Decoupling. A critical observation from Fig. 13(b) is the performance decoupling in the *w/o Supplementary* setting. For Tasks 1, 4, and 5, while the SR remains high (> 0.92), the corresponding *Exec* scores are markedly lower than those in other variants. This indicates that in the absence of grounded environmental supplements, the agent achieves success through inefficient "trial-and-error" rather than informed decision-making.

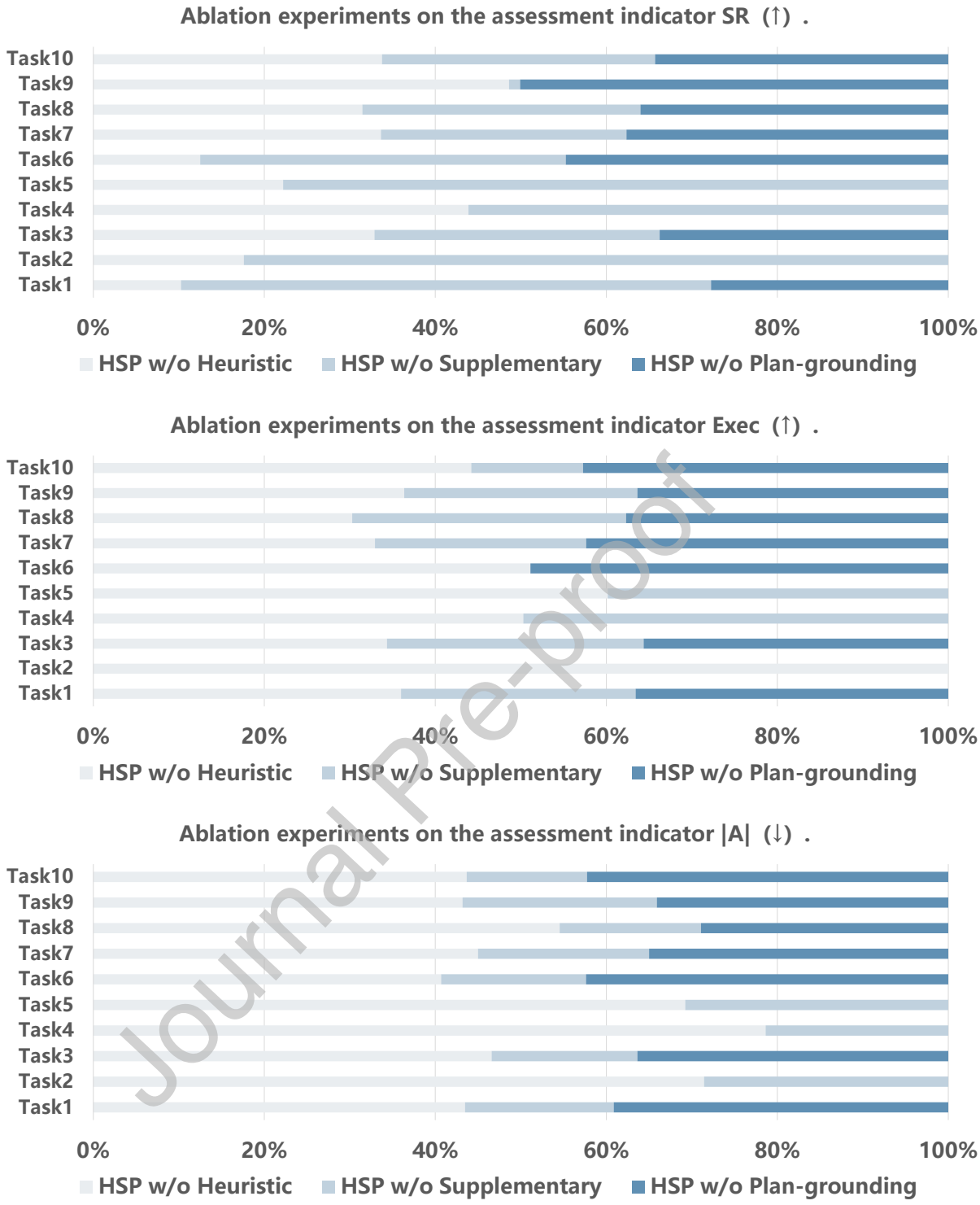

Figure 11: Ablation. HSP ablation experiments with different assessment metrics. In the experiment, we specifically analyzed the roles of different modules in SR, $Exec$, and $|A|$ evaluation metrics.

Criticality of Plan-grounding. The total collapse of SR (0.000) in Tasks 2, 4, and 5 under the *w/o Plan-grounding* variant demonstrates the existence of "grounding-critical" tasks. Unlike other modules whose absence leads to performance degradation, the lack of the P-module results in absolute task failure, proving its role as the fundamental prerequisite for physical interaction.

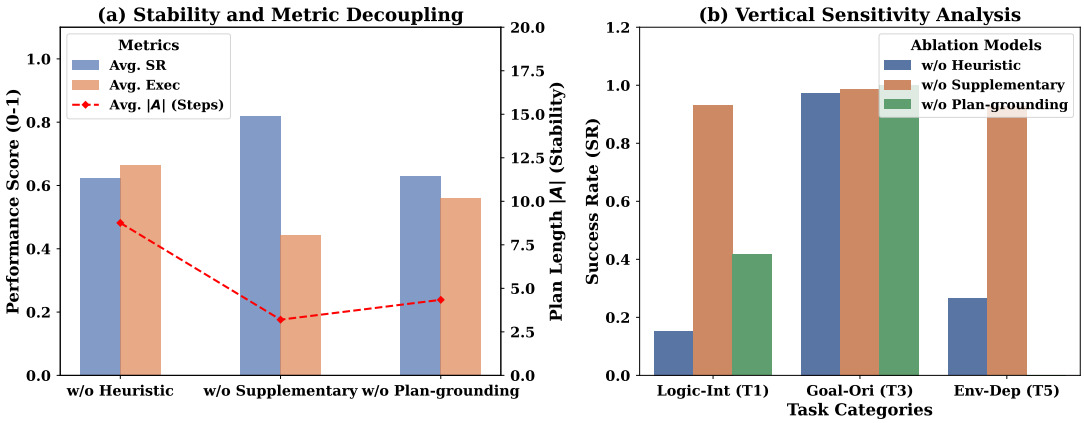

Figure 12: Metric Correlation and Vertical Task Sensitivity in Ablation Studies. (a) **Metric Correlation and Stability**: The disparity between SR and Exec in *w/o Supplementary* highlights the decoupling of logical success and physical execution. The fluctuations in average plan length $|A|$ (red line) for *w/o Heuristic* signify the redundancy and inconsistency in LLM-generated plans without PDDL guidance. (b) **Vertical Task Sensitivity**: Differential performance across task categories. Environment-dependent tasks (T5) show extreme sensitivity to the P-module (Plan-grounding), while Logic-intensive tasks (T1) depend on the H-module (Heuristic). Simple Goal-oriented tasks (T3) remain robust, indicating LLM's baseline reasoning limits.

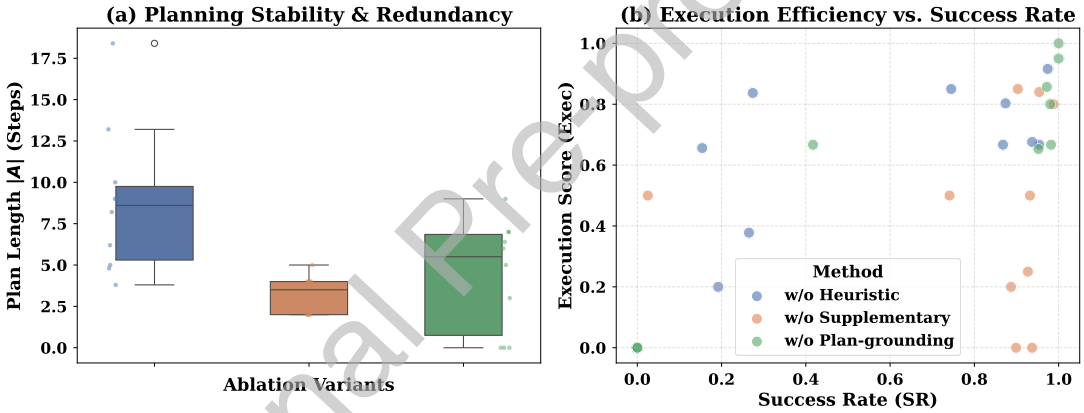

Figure 13: Analysis of Plan Length Distribution and Efficiency in HSP Framework. (a) Distribution of plan length $|A|$, highlighting the stability provided by the H-module. (b) Correlation between Success Rate and Execution Score, revealing the efficiency gap in information-deficient settings.

Findings: Overall, through comparative analysis with baseline frameworks and ablation studies, we find that:

1. *HSP consistently outperforms baseline frameworks across all metrics.* HSP employs PDDL-based heuristic planning to guide LLMs in step completion, while the `Plan-grounding` module ensures reliable task execution.

2. *Every HSP component contributes uniquely to task planning.* The absence of any part affects the agent's planning capability. As the central component of HSP, the `Supplementary` module connects the grounding of heuristic ideas and actions.

3. *LLMs possess only very little planning ability.* As shown in the experimental section, planning using LLM alone is difficult to achieve. This is because LLM is by nature incapable of planning. Despite the concern about the planning ability of LLM, we can utilize LLM as a supplementary tool to help us with planning.

6. Discussion and Limitations

We argue that current LLMs remain deficient in reasoning capabilities, making it difficult to rely solely on them as planners to enhance Agents' performance. As highlighted by [20], LLMs' strengths lie primarily in their text generation

and commonsense knowledge rather than in planning. Furthermore, their Theory of Mind ability [74, 75] prompts reconsideration of LLMs' potential as World Models or Agent Models. In our approach, LLMs supplement and refine planning. Unlike classical methods, our LLM-augmented framework adapts to diverse complex tasks. Compared to LLM-based planners, our PDDL-derived plans increase the success rate. However, our framework has the following **limitations**: 1) *Requirement that the environment be fully observable.* We need to model based on the environment, which makes it difficult to apply our approach to partially observable environments. 2) *PDDL expertise required.* Our framework requires manually supplementing PDDL file generation. 3) *The gap between the simulation environment and the real scenario.* Our framework has not yet been tested in real scenarios and is still in the simulation environment. In real scenarios, many factors still need to be considered, such as physical motion, mechanical control, etc. Our framework is currently intended to contribute to planning, but for real applications, further considerations are needed.

7. Conclusion and Future Work

In this paper, we propose a novel planning framework for HSP. The planning framework comprises three key modules: `Heuristic`, `Supplementary`, and `Plan-grounding`. The `Heuristic` module utilizes the PDDL to generate the planning, which makes up for the LLMs' planning capability deficiency. As for LLMs, they mainly play a role in the `Supplementary` and `Plan-grounding` modules. In the `Supplementary` module, the initial plans generated by the `Heuristic` module are supplemented to make the planning more feasible. The `Plan-grounding` module considers how to convert the natural language into instructions that Agents understand, and at the same time, through interaction with LLMs, it accomplishes error correction during the execution process. We are committed to seeking breakthroughs in the following **challenges**: 1) *Environment speculation based on LLMs.* By utilizing the rich knowledge of LLMs, one-step generation of planning is changed into multi-step planning. To achieve this, the agent must continuously interact with the environment and update LLMs with real-time state and environmental data. Consequently, planning frameworks must support dynamic environment interaction, thereby reducing full observability to partial observability. 2) *Fine-tuning the LLMs.* By fine-tuning the LLMs, the correctness of the LLMs for the generation of PDDL files is improved, thus reducing human involvement in the process of modeling the environment. 3) *Testing in real scenarios.* By incorporating a physical correction module, we hope to integrate real-world physical constraints into our framework, thereby enhancing its capability to execute actions in realistic scenarios.

CRediT authorship contribution statement

Chao Wang: Writing - Original Draft, Writing-Editing, Software, Data curation. **Longhui Cao**: Writing - Original Draft, Writing Editing, Software, Data curation. **Juntong Qi**: Writing -Review, Editing. **Linqi Ye**: Writing -Review, Editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be available upon request.

Acknowledgments

This work was supported by the Natural Science Foundation of Shanghai (No.23ZR1422800).

References

[1] B. Alderson-Day, C. Fernyhough, Inner speech: Development, cognitive functions, phenomenology, and neurobiology., Psychological bulletin 141 (5) (2015) 931.
[2] H. Wang, W. Lin, T. Peng, Q. Xiao, R. Tang, Multi-agent deep reinforcement learning-based approach for dynamic flexible assembly job shop scheduling with uncertain processing and transport times, Expert Systems with Applications (2025) 126441.

[3] J. Shi, J. Zhao, X. Wu, R. Xu, Y.-H. Jiang, L. He, Mitigating reasoning hallucination through multi-agent collaborative filtering, Expert Systems with Applications 263 (2025) 125723.

[4] A. Akakzia, C. Colas, P. Oudeyer, M. Chetouani, O. Sigaud, Grounding language to autonomously-acquired skills via goal generation, in: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021, OpenReview.net, 2021.
URL https://openreview.net/forum?id=chPj_I5KMHG

[5] T. Kurutach, A. Tamar, G. Yang, S. J. Russell, P. Abbeel, Learning plannable representations with causal infogan, in: S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada, 2018, pp. 8747–8758.
URL https://proceedings.neurips.cc/paper/2018/hash/08aac6ac98e59e523995c161e57875f5-Abstract.html

[6] D. Xu, S. Nair, Y. Zhu, J. Gao, A. Garg, L. Fei-Fei, S. Savarese, Neural task programming: Learning to generalize across hierarchical tasks, in: 2018 IEEE international conference on robotics and automation (ICRA), IEEE, 2018, pp. 3795–3802.

[7] B. Eysenbach, R. Salakhutdinov, S. Levine, Search on the replay buffer: Bridging planning and reinforcement learning, in: H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, 2019, pp. 15220–15231.
URL https://proceedings.neurips.cc/paper/2019/hash/5c48ff18e0a47baaf81d8b8ea51eec92-Abstract.html

[8] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, Y. Cao, React: Synergizing reasoning and acting in language models, in: The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023, OpenReview.net, 2023.
URL https://openreview.net/forum?id=WE_vluYUL-X

[9] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al., Chain-of-thought prompting elicits reasoning in large language models, Advances in neural information processing systems 35 (2022) 24824–24837.

[10] H. Zhen, X. Qiu, P. Chen, J. Yang, X. Yan, Y. Du, Y. Hong, C. Gan, 3d-vla: A 3d vision-language-action generative world model, in: Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024, OpenReview.net, 2024.
URL https://openreview.net/forum?id=EZcFK8HupF

[11] J. Lin, Y. Du, O. Watkins, D. Hafner, P. Abbeel, D. Klein, A. D. Dragan, Learning to model the world with language, in: Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024, OpenReview.net, 2024.
URL https://openreview.net/forum?id=7dP6Yq9Uwv

[12] P. Lu, B. Peng, H. Cheng, M. Galley, K. Chang, Y. N. Wu, S. Zhu, J. Gao, Chameleon: Plug-and-play compositional reasoning with large language models, in: A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, S. Levine (Eds.), Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, 2023.
URL http://papers.nips.cc/paper_files/paper/2023/hash/871ed095b734818cfba48db6aeb25a62-Abstract-Conference.html

[13] H. Huang, B. Xu, X. Liang, K. Chen, M. Yang, T. Zhao, C. Zhu, Multi-view fusion for instruction mining of large language model, Information Fusion 110 (2024) 102480. doi:https://doi.org/10.1016/j.inffus.2024.102480.
URL https://www.sciencedirect.com/science/article/pii/S1566253524002586

[14] C. H. Song, B. M. Sadler, J. Wu, W. Chao, C. Washington, Y. Su, Llm-planner: Few-shot grounded planning for embodied agents with large language models, in: IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023, IEEE, 2023, pp. 2986–2997. doi:10.1109/ICCV51070.2023.00280.
URL https://doi.org/10.1109/ICCV51070.2023.00280

[15] Y. Hu, F. Lin, T. Zhang, L. Yi, Y. Gao, Look before you leap: Unveiling the power of GPT-4V in robotic vision-language planning, CoRR abs/2311.17842. arXiv:2311.17842, doi:10.48550/ARXIV.2311.17842.
URL https://doi.org/10.48550/arXiv.2311.17842

[16] X. Yan, Y. Song, X. Cui, F. Christianos, H. Zhang, D. H. Mguni, J. Wang, Ask more, know better: Reinforce-learned prompt questions for decision making with large language models, CoRR abs/2310.18127. arXiv:2310.18127, doi:10.48550/ARXIV.2310.18127.
URL https://doi.org/10.48550/arXiv.2310.18127

[17] X. Chen, S. Zhang, P. Zhang, L. Zhao, J. Chen, Asking before acting: Gather information in embodied decision making with language models, arXiv preprint arXiv:2305.15695.

[18] S. Hao, T. Liu, Z. Wang, Z. Hu, Toolkengpt: Augmenting frozen language models with massive tools via tool embeddings, in: A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, S. Levine (Eds.), Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, 2023.
URL http://papers.nips.cc/paper_files/paper/2023/hash/8fd1a81c882cd45f64958da6284f4a3f-Abstract-Conference.html

[19] Z. Liu, A. Bahety, S. Song, REFLECT: summarizing robot experiences for failure explanation and correction, in: J. Tan, M. Toussaint, K. Darvish (Eds.), Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA, Vol. 229 of Proceedings of Machine Learning Research, PMLR, 2023, pp. 3468–3484.
URL https://proceedings.mlr.press/v229/liu23g.html

[20] S. Kambhampati, K. Valmeekam, L. Guan, K. Stechly, M. Verma, S. Bhambri, L. Saldyt, A. Murthy, Llms can't plan, but can help planning in llm-modulo frameworks, arXiv preprint arXiv:2402.01817.

[21] S. Hao, Y. Gu, H. Ma, J. J. Hong, Z. Wang, D. Z. Wang, Z. Hu, Reasoning with language model is planning with world model, in: H. Bouamor, J. Pino, K. Bali (Eds.), Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023, Association for Computational Linguistics, 2023, pp. 8154–8173. doi:10.18653/V1/2023.EMNLP-MAIN.507.
URL https://doi.org/10.18653/v1/2023.emnlp-main.507

[22] K. Valmeekam, M. Marquez, S. Sreedharan, S. Kambhampati, On the planning abilities of large language models-a critical investigation, Advances in Neural Information Processing Systems 36 (2023) 75993–76005.

[23] IPC, International planning competition (1998).
URL https://www.icaps-conference.org/competitions/

[24] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, Y. Iwasawa, Large language models are zero-shot reasoners, in: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022, 2022.
URL http://papers.nips.cc/paper_files/paper/2022/hash/8bb0d291acd4acf06ef112099c16f326-Abstract-Conference.html

[25] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. V. Le, E. H. Chi, Least-to-most prompting enables complex reasoning in large language models, in: The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023, OpenReview.net, 2023.
URL https://openreview.net/forum?id=WZH7099tgfM

[26] M. Shridhar, X. Yuan, M. Côté, Y. Bisk, A. Trischler, M. J. Hausknecht, Alfworld: Aligning text and embodied environments for interactive learning, in: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021, OpenReview.net, 2021.
URL https://openreview.net/forum?id=0IOX0YcCdTn

[27] K. Stechly, M. Marquez, S. Kambhampati, GPT-4 doesn't know it's wrong: An analysis of iterative prompting for reasoning problems, CoRR abs/2310.12397. arXiv:2310.12397, doi:10.48550/ARXIV.2310.12397.
URL https://doi.org/10.48550/arXiv.2310.12397

[28] Z. Zhao, S. Song, B. Duah, J. C. Macbeth, S. A. Carter, M. P. Van, N. S. Bravo, M. Klenk, K. A. Sieck, A. L. S. Filipowicz, More human than human: Llm-generated narratives outperform human-llm interleaved narratives, in: Creativity and Cognition, C&C 2023, Virtual Event, USA, June 19-21, 2023, ACM, 2023, pp. 368–370. doi:10.1145/3591196.3596612.
URL https://doi.org/10.1145/3591196.3596612

[29] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, G. Lample, Llama: Open and efficient foundation language models, CoRR abs/2302.13971. arXiv:2302.13971, doi:10.48550/ARXIV.2302.13971.
URL https://doi.org/10.48550/arXiv.2302.13971

[30] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, K. Narasimhan, Tree of thoughts: Deliberate problem solving with large language models, in: A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, S. Levine (Eds.), Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, 2023.
URL http://papers.nips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html

[31] K. Zhang, F. Zhou, L. Wu, N. Xie, Z. He, Semantic understanding and prompt engineering for large-scale traffic data imputation, Information Fusion 102 (2024) 102038. doi:https://doi.org/10.1016/j.inffus.2023.102038.
URL https://www.sciencedirect.com/science/article/pii/S1566253523003548

[32] L. Guan, K. Valmeekam, S. Sreedharan, S. Kambhampati, Leveraging pre-trained large language models to construct and utilize world models for model-based task planning, Advances in Neural Information Processing Systems 36 (2023) 79081–79094.

[33] W. Huang, P. Abbeel, D. Pathak, I. Mordatch, Language models as zero-shot planners: Extracting actionable knowledge for embodied agents, in: International conference on machine learning, PMLR, 2022, pp. 9118–9147.

[34] Z. Zhao, W. S. Lee, D. Hsu, Large language models as commonsense knowledge for large-scale task planning, Advances in Neural Information Processing Systems 36.

[35] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, A. Garg, Progprompt: Generating situated robot task plans using large language models, in: 2023 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2023, pp. 11523–11530.

[36] C. Ming, J. Lin, P. Fong, H. Wang, X. Duan, J. He, Hicrisp: A hierarchical closed-loop robotic intelligent self-correction planner, CoRR abs/2309.12089. arXiv:2309.12089, doi:10.48550/ARXIV.2309.12089.
URL https://doi.org/10.48550/arXiv.2309.12089

[37] J. Hoffmann, Ff: The fast-forward planning system, AI magazine 22 (3) (2001) 57–57.

[38] M. Helmert, The fast downward planning system, Journal of Artificial Intelligence Research 26 (2006) 191–246.

[39] D. Bryce, S. Kambhampati, A tutorial on planning graph based reachability heuristics, AI Magazine 28 (1) (2007) 47–47.

[40] C. R. Garrett, T. Lozano-Pérez, L. P. Kaelbling, Pddlstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning, in: Proceedings of the international conference on automated planning and scheduling, Vol. 30, 2020, pp. 440–448.

[41] Y.-q. Jiang, S.-q. Zhang, P. Khandelwal, P. Stone, Task planning in robotics: an empirical comparison of pddl-and asp-based systems, Frontiers of Information Technology & Electronic Engineering 20 (2019) 363–373.

[42] T. Silver, R. Chitnis, N. Kumar, W. McClinton, T. Lozano-Perez, L. P. Kaelbling, J. Tenenbaum, Inventing relational state and action abstractions for effective and efficient bilevel planning, arXiv preprint arXiv:2203.09634.

[43] L. Cao, C. Wang, J. Qi, Y. Peng, Exploring into the unseen: Enhancing language-conditioned policy generalization with behavioral information, Cyborg and Bionic Systems 5 (2024) 0084.

[44] Z. Xiao, P. Li, C. Liu, H. Gao, X. Wang, Macns: A generic graph neural network integrated deep reinforcement learning based multi-agent collaborative navigation system for dynamic trajectory planning, Information Fusion 105 (2024) 102250. doi:https://doi.org/10.1016/j.inffus.2024.102250.

URL https://www.sciencedirect.com/science/article/pii/S1566253524000289

[45] S. Nahavandi, R. Alizadehsani, D. Nahavandi, C. P. Lim, K. Kelly, F. Bello, Machine learning meets advanced robotic manipulation, Information Fusion 105 (2024) 102221. doi:https://doi.org/10.1016/j.inffus.2023.102221.
URL https://www.sciencedirect.com/science/article/pii/S1566253523005377

[46] H. Feng, Q. Li, W. Wang, A. K. Bashir, A. K. Singh, J. Xu, K. Fang, Security of target recognition for uav forestry remote sensing based on multi-source data fusion transformer framework, Information Fusion 112 (2024) 102555. doi:https://doi.org/10.1016/j.inffus.2024.102555.
URL https://www.sciencedirect.com/science/article/pii/S1566253524003336

[47] D. Xu, R. Martín-Martín, D.-A. Huang, Y. Zhu, S. Savarese, L. F. Fei-Fei, Regression planning networks, Advances in neural information processing systems 32.

[48] D. Shah, P. Xu, Y. Lu, T. Xiao, A. Toshev, S. Levine, B. Ichter, Value function spaces: Skill-centric state abstractions for long-horizon reasoning, in: The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022, OpenReview.net, 2022.
URL https://openreview.net/forum?id=vgqS1vkkCbE

[49] J. Chen, S. Yuan, R. Ye, B. P. Majumder, K. Richardson, Put your money where your mouth is: Evaluating strategic planning and execution of llm agents in an auction arena, arXiv preprint arXiv:2310.05746.

[50] M. Hu, Y. Mu, X. Yu, M. Ding, S. Wu, W. Shao, Q. Chen, B. Wang, Y. Qiao, P. Luo, Tree-planner: Efficient close-loop task planning with large language models, in: The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024, OpenReview.net, 2024.
URL https://openreview.net/forum?id=Glcsog6zOe

[51] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, et al., A survey on large language model based autonomous agents, Frontiers of Computer Science 18 (6) (2024) 186345.

[52] Y. Zhang, C. Wang, J. Qi, Y. Peng, Leave it to large language models! correction and planning with memory integration, Cyborg and Bionic Systems 5 (2024) 0087.

[53] T. Silver, S. Dan, K. Srinivas, J. B. Tenenbaum, L. Kaelbling, M. Katz, Generalized planning in pddl domains with pretrained large language models, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 38, 2024, pp. 20256–20264.

[54] Z. Wang, S. Cai, G. Chen, A. Liu, X. Ma, Y. Liang, Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents, arXiv preprint arXiv:2302.01560.

[55] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, L. Fei-Fei, Voxposer: Composable 3d value maps for robotic manipulation with language models, in: J. Tan, M. Toussaint, K. Darvish (Eds.), Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA, Vol. 229 of Proceedings of Machine Learning Research, PMLR, 2023, pp. 540–562.
URL https://proceedings.mlr.press/v229/huang23b.html

[56] J. Duan, S. Yu, H. L. Tan, H. Zhu, C. Tan, A survey of embodied ai: From simulators to research tasks, IEEE Transactions on Emerging Topics in Computational Intelligence 6 (2) (2022) 230–244.

[57] R. Bairi, A. Sonwane, A. Kanade, A. Iyer, S. Parthasarathy, S. Rajamani, B. Ashok, S. Shet, Codeplan: Repository-level coding using llms and planning, Proceedings of the ACM on Software Engineering 1 (FSE) (2024) 675–698.

[58] T. Silver, V. Hariprasad, R. S. Shuttleworth, N. Kumar, T. Lozano-Pérez, L. P. Kaelbling, Pddl planning with pretrained large language models, in: NeurIPS 2022 foundation models for decision making workshop, 2022.

[59] M. Zhang, O. Press, W. Merrill, A. Liu, N. A. Smith, How language model hallucinations can snowball, arXiv preprint arXiv:2305.13534.

[60] K. Lin, C. Agia, T. Migimatsu, M. Pavone, J. Bohg, Text2motion: From natural language instructions to feasible plans, Autonomous Robots 47 (8) (2023) 1345–1365.

[61] Q. Cheng, T. Sun, X. Liu, W. Zhang, Z. Yin, S. Li, L. Li, K. Chen, X. Qiu, Can ai assistants know what they don't know?, arXiv preprint arXiv:2401.13275.

[62] Z. Yang, G. Chen, X. Li, W. Wang, Y. Yang, Doraemongpt: Toward understanding dynamic scenes with large language models, arXiv preprint arXiv:2401.08392.

[63] J. Xiang, T. Tao, Y. Gu, T. Shu, Z. Wang, Z. Yang, Z. Hu, Language models meet world models: Embodied experiences enhance language models, Advances in neural information processing systems 36.

[64] A. Barbin, F. Cerutti, A. E. Gerevini, Learning reliable pddl models for classical planning from visual data, in: 2024 IEEE 36th International Conference on Tools with Artificial Intelligence (ICTAI), IEEE, 2024, pp. 722–728.

[65] X. Zhang, H. Qin, F. Wang, Y. Dong, J. Li, Lamma-p: Generalizable multi-agent long-horizon task allocation and planning with lm-driven pddl planner, arXiv preprint arXiv:2409.20560.

[66] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, P. Stone, LLM+P: empowering large language models with optimal planning proficiency, CoRR abs/2304.11477. arXiv:2304.11477, doi:10.48550/ARXIV.2304.11477.
URL https://doi.org/10.48550/arXiv.2304.11477

[67] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, et al., A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions, arXiv preprint arXiv:2311.05232.

[68] H. Ye, T. Liu, A. Zhang, W. Hua, W. Jia, Cognitive mirage: A review of hallucinations in large language models, arXiv preprint arXiv:2309.06794.

[69] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford, et al., Gpt-4o system card, arXiv preprint arXiv:2410.21276.

[70] S. Bai, K. Chen, X. Liu, J. Wang, W. Ge, S. Song, K. Dang, P. Wang, S. Wang, J. Tang, H. Zhong, Y. Zhu, M. Yang, Z. Li, J. Wan, P. Wang, W. Ding, Z. Fu, Y. Xu, J. Ye, X. Zhang, T. Xie, Z. Cheng, H. Zhang, Z. Yang, H. Xu, J. Lin, Qwen2.5-vl technical report, arXiv preprint arXiv:2502.13923.

[71] X. Sun, Y. Chen, Y. Huang, R. Xie, J. Zhu, K. Zhang, S. Li, Z. Yang, J. Han, X. Shu, et al., Hunyuan-large: An open-source moe model with 52 billion activated parameters by tencent, arXiv preprint arXiv:2411.02265.
[72] Baidu-ERNIE-Team, Ernie 4.5 technical report (2025).
[73] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, et al., The llama 3 herd of models, arXiv preprint arXiv:2407.21783.
[74] T. Ullman, Large language models fail on trivial alterations to theory-of-mind tasks, arXiv preprint arXiv:2302.08399.
[75] M. Kosinski, Theory of mind might have spontaneously emerged in large language models, arXiv preprint arXiv:2302.02083.