Robotics and Autonomous Systems 200 (2026) 105404

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Onto-LLM-TAMP: Knowledge-oriented Task and Motion Planning using Large Language Models

Muhayy Ud Din [a], Jan Rosell [b], Waseem Akram [a], Isiah Zaplana [b], Maximo A. Roa [c], Irfan Hussain [a,*]

[a] Khalifa University Center for Autonomous Robotic Systems (KUCARS), Khalifa University, United Arab Emirates
[b] Institute of Industrial and Control Engineering (IOC), Universitat Politecnica de Catalunya, Spain
[c] Institute of Robotics and Mechatronics, German Aerospace Center (DLR), Germany

ARTICLE INFO

Keywords:
Task and motion planning
Large Language Models
Ontological knowledge
Reasoning

ABSTRACT

Performing complex manipulation tasks in dynamic environments requires efficient Task and Motion Planning (TAMP) approaches that combine high-level symbolic plans with low-level motion control. Advances in Large Language Models (LLMs), such as GPT-4, are transforming task planning by offering natural language as an intuitive and flexible way to describe tasks, generate symbolic plans, and reason. However, the effectiveness of LLM-based TAMP approaches is limited due to static and template-based prompting, which limits adaptability to dynamic environments and complex task contexts. To address these limitations, this work proposes a novel Onto-LLM-TAMP framework that employs knowledge-based reasoning to refine and expand user prompts with task-contextual reasoning and knowledge-based environment state descriptions. Integrating domain-specific knowledge into the prompt ensures semantically accurate and context-aware task plans. The proposed framework demonstrates its effectiveness by resolving semantic errors in symbolic plan generation, such as maintaining logical temporal goal ordering in scenarios involving hierarchical object placement. The proposed framework is validated through both simulation and real-world scenarios, demonstrating significant improvements over the baseline approach in terms of adaptability to dynamic environments and the generation of semantically correct task plans. Videos and code can be found here: https://muhayyuddin.github.io/llm-tamp/.

1. Introduction

Task and motion planning (TAMP) is a fundamental capability for robotic manipulation in complex environments, as it enables robots to combine high-level symbolic reasoning with low-level geometric feasibility. In a typical TAMP pipeline, the task planner generates an abstract sequence of actions to achieve a desired goal, while the motion planner computes collision-free trajectories that respect physical and kinematic constraints. Classical TAMP solutions are based on formal planning representations, such as the Planning Domain Description Language (PDDL) [1], together with symbolic planners such as GraphPlan [2] or FastForward [3]. The resulting symbolic plans are then executed by invoking a motion planner for each action, with a dedicated interface coordinating task-level and motion-level reasoning.

Although PDDL-based TAMP has proven effective in structured domains, it presents notable limitations in practical settings. Task specifications require manual modeling of objects, actions, and constraints, which becomes impractical in dynamic or unstructured environments. Moreover, the syntax and abstraction level of PDDL can be difficult for non-expert users to adopt [4]. In contrast, natural language offers a more intuitive and flexible interface for task specification, allowing users to express goals without requiring explicit formalization [5]. Motivated by this, recent advances in large language models (LLMs), such as GPT-4 [6], have accelerated research on natural-language-driven task planning [7] and task and motion planning [8,9].

LLM-based TAMP approaches typically employ pre-trained models, such as GPT-4, to translate natural language task descriptions into symbolic action sequences [9]. These models have demonstrated strong reasoning capabilities, particularly when supported by in-context prompting [10]. However, the quality of the generated plans is highly sensitive to prompt design. Weak or underspecified prompts can significantly degrade planning performance, motivating recent work on prompt engineering for better task elaboration. For instance, fixed template-based prompts are used in [9] to describe spatial and geometric relations, while [8] employs separate templates for scene description and environment state textualization. Interactive prompting strategies have

* Corresponding author.
E-mail address: irfan.hussain@ku.ac.ae (I. Hussain).

https://doi.org/10.1016/j.robot.2026.105404
Received 2 October 2025; Received in revised form 17 January 2026; Accepted 24 February 2026
Available online 5 March 2026

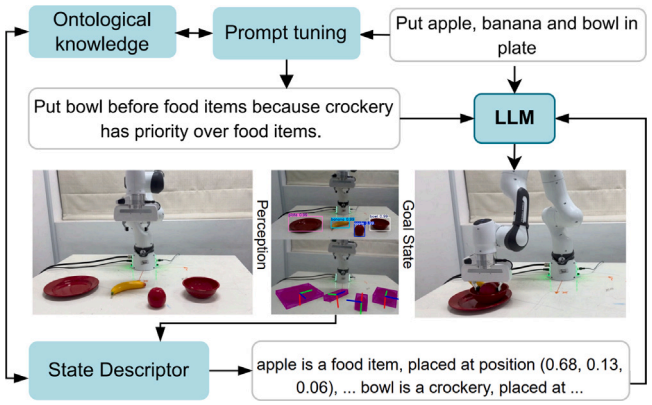

Fig. 1. Task execution using the Onto-LLM-TAMP approach. The prompt is processed through the Prompt Tuning module, which uses knowledge-based reasoning to elaborate the user prompt. The state of the environment is textualized using ontological knowledge by the State Descriptor. This detailed elaboration enables the LLM to generate a semantically correct symbolic plan.

also been explored, in which user input is combined with stored task guidelines to refine planning instructions [11].

Despite these advances, most of the existing prompt generation methods rely on static templates. Such prompts tend to be repetitive and inflexible, limiting their ability to capture the evolving context of dynamic environments. As a result, they often fail to generalize to new task formulations or unexpected situations. Moreover, template-based prompts may oversimplify complex environment states, leading to incomplete or misleading task descriptions and, consequently, semantically incorrect plans. Since effective prompting is critical for reliable LLM-based planning [12], there is a clear need for more adaptive and context-aware prompt engineering strategies.

To address these limitations, this work proposes an *Onto-LLM-TAMP* approach that used knowledge-based reasoning to refine and expand user prompts within the task context. Building on prior LLM-based task and motion planning frameworks, such as LLM-TAMP [9], we introduce an ontology-driven prompt tuning mechanism that augments natural language instructions with task-relevant semantic constraints.

For example, consider the scenario shown in Fig. 1, where the user instructs the robot to *"put the banana, apple, and bowl in the plate"*. Without explicit semantic constraints, LLM-based task planners may generate a plan that first places the food items and then places the bowl. While syntactically valid, such a plan is semantically incorrect, as it may result in placing the bowl on top of the food items. The proposed Onto-LLM-TAMP framework addresses this issue by incorporating domain-specific knowledge into the prompt, such as the rule that crockery should be placed before food items. By supplementing the prompt with such ontological constraints, the LLM is guided to generate semantically consistent and logically ordered task plans.

Although recent studies have also explored the integration of ontological knowledge with LLM-based task planning [13], the role of ontology in these approaches differs fundamentally from the one adopted in this work. Existing ontology-enhanced planners primarily use object-level attributes and affordances to refine low-level manipulation decisions, such as grasp selection or force modulation during execution (e.g. `glass → pick gently`, `wood → pick stably`). In contrast, the proposed Onto-LLM-TAMP framework employs ontological reasoning at the task level to explicitly guide prompt construction for LLM-based planning. Rather than modifying the action primitives or execution policies, the ontology is used to encode semantic prioritization and temporal ordering constraints, which are incorporated into the prompt to guide the LLM toward generating semantically consistent and logically ordered symbolic plans. This distinction enables the proposed approach to address planning failures that arise from incorrect task-level reasoning, such as flawed temporal goal ordering, which are not explicitly targeted by prior ontology-based LLM planners.

Contributions

The core contribution of this work is a novel Onto-LLM-TAMP framework for task and motion planning that enables LLMs to generate semantically accurate and context-aware plans. Building upon this framework, the main contributions are:

- *Automated prompt tuning with task contextual reasoning:* Task-relevant commonsense and procedural knowledge is extracted from an ontology to guide LLMs toward logically ordered and semantically correct action sequences.
- *Knowledge-based environment state description:* Ontological reasoning is used to construct detailed, context-aware descriptions of the environment, improving the effectiveness of prompt-based task planning.
- *Benchmarking of LLM models:* The proposed approach is evaluated across multiple LLMs, including GPT, LLaMA, Gemini, and Cohere, demonstrating the impact of ontology-driven prompt tuning on planning performance.

The remainder of the paper is organized as follows. Section 2 reviews related work on task and motion planning and LLM-based planning approaches. Section 3 formulates the problem and presents the Onto-LLM-TAMP framework. Section 4 details the proposed methodology, ontology integration, and execution pipeline. Experimental results in simulation and real-world settings are presented in Section 5. Finally, Section 6 concludes the paper.

2. Related work

2.1. Task and motion planning

TAMP uses a structured approach where task planning and motion planning are handled separately but in a coordinated manner. Task planning typically uses symbolic AI techniques, such as hierarchical planning, to generate a sequence of discrete actions based on predefined rules or logic. Motion planning uses geometric algorithms, such as sampling-based planners, to compute feasible paths [14,15]. A declarative language, such as PDDL, is used for TAMP. However, its use is limited due to the requirement of a large number of parameters to define the problem, actions, and effects [16]. Moreover, as the action space expands, finding geometrically feasible symbolic action sequences becomes computationally challenging without effective heuristics [17]. Recent studies have explored data-driven heuristics to enhance TAMP efficiency [18,19]. Other approaches use knowledge-based reasoning to reduce the action space, using domain-specific ontological knowledge, and then apply heuristics for efficient symbolic plan generation [20,21]. The approach proposed here is a general manipulation framework to work across various domains, with the specific application ontology providing the specialized knowledge needed to correctly solve the task.

2.2. Large language models

Large models represent a significant advancement in artificial intelligence, enabling powerful language understanding and general reasoning capabilities in various domains. Several models have been proposed including GPT-4 [22], Gemini [23], LLaMA [24], Cohere Language Models [25], and DeepSeek [26]. These models are built on transformer architectures trained on vast datasets. GPT-4 excels in complex reasoning and code synthesis, while Gemini integrates multimodal capabilities, allowing it to process text, images, and video. LLaMA stands out as an efficient open-source model that facilitates research and development by the broader community. Cohere offers industry-focused models that are optimized for industry-related applications. DeepSeek is another emerging model that aims to bridge the performance gap between open-source models and cutting-edge proprietary systems, focusing on multilingual and domain-specific tasks.

2.3. LLMs for TAMP

These foundation models are increasingly being used in task planning and symbolic reasoning for robotics and automated systems, i.e., LLMs are being explored to enhance task planning by using their ability to understand and generate human-like instructions, descriptions, and planning sequences. In particular, they allow to translate natural language task descriptions into formal task representations without the need to manually represent the problem domain [27,28]. Due to their extensive language understanding and contextual reasoning capabilities, LLMs are used to generate high-level task plans and their translation into feasible motion sequences with minimal additional training [29,30].

For example, GPT-4 has been adapted for robotic task planning by generating high-level action sequences and providing natural language explanations for task execution [31]. Google's SayCan system combines LLMs with affordance-based planning to ground language commands into feasible robot actions [29]. Similarly, language models have been explored as zero-shot planners, enabling agents to extract actionable plans directly from textual instructions [32]. These approaches demonstrate the potential of foundation models to serve as cognitive reasoning engines in task and motion planning (TAMP). For instance, LLM-GROP [8] utilizes a pre-trained LLM to define symbolic goals for object placement and rearrangement, integrating these with traditional TAMP methods, and AutoTAMP [4] employs a pre-trained LLM to convert user inputs into formal language, which is then used in a TAMP algorithm.

In all the approaches mentioned above, effective prompting techniques are crucial to optimize LLM performance [12]. In addition, [31] identified key challenges in prompting LLMs for robotic manipulation, including the need for detailed and accurate problem descriptions, as well as managing biases that shape response structures. This study tackles the issue of effective prompting for task and motion planning by utilizing ontological knowledge to enhance task elaboration for LLM.

3. Problem formulation

This section formally defines the task and motion planning problem and how we model the problem for Onto-LLM-TAMP.

3.1. Task and motion planning

A task and motion planning problem is typically described as a tuple $\mathcal{X} = \langle \mathcal{O}, \mathcal{S}, \mathcal{A}, \mathcal{T}, s_0, g \rangle$, where:

- $\mathcal{O}$ denotes the set of objects in the environment.
- $\mathcal{S}$ represents the state space of all possible configurations of objects within the environment, where $s_t \in \mathcal{S}$ signifies the state of the objects at time t.
- $\mathcal{A}$ is the set of primitive actions, each one requiring the call to the motion planner for its execution. A primitive action $a \in \mathcal{A}$ is defined by a set of objects $o \in \mathcal{O}$ and a set of continuous parameters γ, e.g., `Place(cup, [x, y, z, θ])`. The parameters γ serve as goals for the motion planner and a is considered feasible if the motion planner finds a solution, i.e., a collision-free trajectory τ. We denote such a feasible action as $a(\tau)$.
- $\mathcal{T}$ is the state transition function that generates the subsequent state s_{t+1} after executing an action $a_t(\tau_t)$ at state s_t, formally represented as $s_{t+1} = \mathcal{T}(s_t, a_t(\tau_t))$. This transition function can be evaluated with a black-box simulator.
- s_0 is the initial state, with $s_0 \in \mathcal{S}$.
- g is the goal function, defined as $g(s) : \mathcal{S} \to \{0, 1\}$, which checks if the task goal has been met at a given state s.

The objective in Task and Motion Planning is to derive a sequence of feasible actions $\mathbf{a} = \{a_0(\tau_0), a_1(\tau_1), \ldots, a_T(\tau_T)\}$, such that $s_{t+1} = \mathcal{T}(s_t, a_t(\tau_t))$, for each $t = 0, 1, \ldots, T$, and $g(s_{T+1}) = 1$.

3.2. Problem modeling

The Onto-LLM-TAMP framework is defined as a tuple $\langle \mathcal{P}, \mathcal{K}, \xi, \mathcal{X}, \mathcal{L} \rangle$, where each component plays a crucial role in structuring a feasible action plan:

- $\mathcal{P}$ represents the prompt, containing essential information such as the task description, environment state, and details about the objects involved.
- $\mathcal{K}$ denotes the ontological knowledge-base, which provides domain-specific information and task-related rules.
- ξ is the knowledge reasoner, responsible for enriching the prompt with task-specific correctness knowledge:

$$\xi : \mathcal{P} \times \mathcal{K} \to \epsilon \tag{1}$$

  where ϵ represents additional guidance relevant to the task.
- $\mathcal{X}$ is the set of task parameters.
- $\mathcal{L}$ is the LLM model, such as GPT-4, that operates as a black-box component, defined by:

$$\mathcal{L} : \mathcal{P} \times \epsilon \times \mathcal{X} \to \mathbf{a} \tag{2}$$

This model receives the prompt $\mathcal{P}$, enhanced task knowledge ϵ, and TAMP parameters $\mathcal{X}$, and it outputs a set of feasible actions $\mathbf{a} \subset \mathcal{A}$ that enable the robot to transition from the initial environment state to the desired goal state.

The effectiveness of $\mathcal{L}$ depends significantly on the quality of the prompt, since an elaborate and semantically accurate prompt improves the probability of generating a valid sequence of actions. The primary objective of this work is to explore how ontological knowledge and semantic reasoning can further refine and enrich prompts, thereby enhancing the accuracy of semantically correct action sequence generation for consistent task execution.

4. Proposed approach

This section explains how ontological knowledge is used to generate suggestions for the LLM, ensuring semantically correct symbolic plan generation with proper temporal goal ordering. The architecture of our Onto-LLM-TAMP framework is illustrated in Fig. 2. It is composed of *Ontological Prompt Construction Layer* and the *Planning Layer*.

The *Ontological Prompt Construction Layer* is responsible for creating a prompt to be sent to the LLM-based planning module. This is achieved by using user input, predefined prompt templates, and the information available in an ontology. It comprises six modules: (1) the *Ontology* module that contains the application ontology, comprising classes, properties, and rules (detailed in Section 4.1); (2) the *Semantic Tagging* module that analyzes the user input and facilitates the queries to the ontology (discussed in Section 4.2); (3) the *Contextual Inference Engine* module that, using inferred knowledge composed of priorities and rules based on the relationships between objects and actions, suggests the right order of actions in the plan to perform the task in a semantically and logically correct manner (explained in Section 4.3); (4) the *Perception Module* that identifies all objects within the environment, estimating their spatial positions and orientations using YOLO [33] and FoundationPose [34], and instantiating them in the ontology, as done in a similar way in [35]; (5) the *Environmental State Descriptor* module that translates the environmental state retrieved from the ontology into a text format, describing the spatial layout and types of objects in the environment (the perception module and the environment state descriptor are detailed in Section 4.4); (6) The *Prompt Generator* module, that combines the prompt templates with the information coming from the *Contextual Inference Engine* and *Environmental State Descriptor* modules (explained in Section 4.5).

The *Planning Layer* takes a prompt as input and generates a plan to execute. It closely follows the baseline approach [9] and comprises

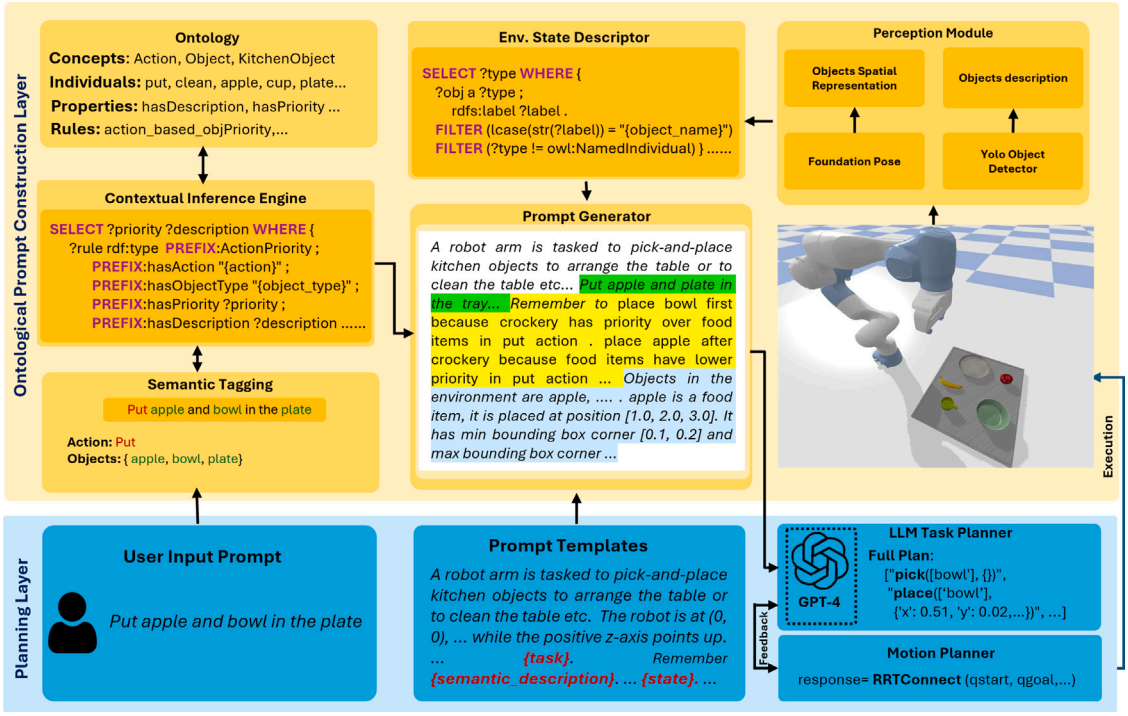

Fig. 2. The Onto-LLM-TAMP framework enhances prompt elaboration for generating semantically accurate symbolic plans. User instructions are first processed through semantic tagging to extract actions and objects. The Contextual Inference Engine queries ontology using SPARQL to retrieve object types, priorities, and task-level semantic constraints that guide correct action sequencing. The Perception Module detects objects using a YOLO-based model and estimates their 6D poses using FoundationPose, providing real-time spatial information. The Environment State Descriptor employs SPARQL queries to classify detected objects into semantic categories defined in the ontology. The resulting semantic and spatial information is textualized and passed to the Prompt Generator, which produces an enriched prompt for the LLM Task Planner to generate a structured task plan. Finally, the Motion Planner executes the task by computing feasible, collision-free trajectories. .

three modules: (1): the *User Input* which describes the task to be performed; (2) the *Prompt Template* which contains the static sections of the prompt, provides general task descriptions and structures the overall prompt[1]; (3) the *LLM-TAMP Module*, detailed in Section 4.6, that has the *LLM Task Planner* to compute a symbolic plan, and a motion planner to compute the motions for each action of the plan.

4.1. Ontological knowledge representation

We developed a simple kitchen domain ontology to provide semantic reasoning for performing tasks in a kitchen environment. This domain-specific ontology enhances reasoning about task constraints, priority rules and object relations, and for this, three main classes have been defined: `Task`, `ActionPriority` and `Object`. The `Object` class is further subdivided into six subclasses describing different kitchen object types: `FoodItems`, `BoxedFood`, `Crockery`, `Utensil`, `KitchenItems`, and `Container`. The hierarchy of classes, object properties, data properties, and some example individuals are shown in Fig. 3.

The data properties of the ontology describe the essential attributes of different objects, including action priority values and specific descriptions with temporal constraints, such as `pick after FoodItem` or `place Crockery before FoodItem`. These descriptions are utilized by the reasoner for prompt tuning, enabling it to guide action sequencing. Other data properties capture object-specific attributes like position, orientation, bounding box, and additional task-related properties. Object properties, on the other hand, define relationships between objects and actions. For instance, `has object type` specifies the

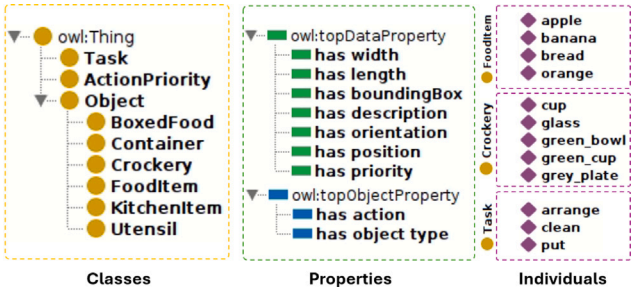

Fig. 3. Kitchen ontology, showing the hierarchy of Classes (yellow), Properties (green and blue), and Individuals (purple). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

classification of an object (e.g., FoodItem or Crockery), while `has action` associates relevant actions, such as "put", "clean" or "arrange" with each object. These properties are essential for knowledge-based reasoning about task execution.

We define a set of rules[2] that specify the correct order of action sequences when manipulating objects in the kitchen environment. These rules are implemented as instances of `ActionPriority`, where each rule associates a task (such as `put` or `clean`) with an object type (such as `FoodItem` or `Crockery`) and assigns a priority level. Through these rules, the reasoning mechanism infers a semantically correct action sequence for task execution. For example, Eq. (3) and Eq. (4)

[1] Complete template can be found here: https://github.com/Muhayyuddin/llm-tamp/blob/llm-site/assets/prompt/txt.

[2] Complete rule set used in this study is available here: https://github.com/Muhayyuddin/llm-tamp/blob/llm-site/assets/smart_kitchen.rdf.

show the description logic representation for two rules, named Rule 1 and Rule 2.

$$\text{Rule1} \sqsubseteq \text{ActionPriority} \sqcap \exists \text{hasAction}.\text{Put} \sqcap \exists \text{hasObjectType}.\text{Crockery} \sqcap \exists \text{hasPriority}.1 \quad (3)$$

$$\text{Rule2} \sqsubseteq \text{ActionPriority} \sqcap \exists \text{hasAction}.\text{Put} \sqcap \exists \text{hasObjectType}.\text{FoodItem} \sqcap \exists \text{hasPriority}.2 \quad (4)$$

These rules define priority constraints for performing the "Put" action on different object types. Rule 1 establishes that when placing objects, items classified as Crockery should have the highest priority, meaning they should be placed before other object types. This prioritization is explicitly set by assigning a priority value of 1 to Crockery, ensuring that these objects are handled first in the sequence of actions.

Similarly, Rule 2 specifies that objects that belong to the FoodItem category have a lower priority when performing the same "Put" action. With a priority value of 2, FoodItems are placed after Crockery, reflecting a structured ordering in object manipulation. These rules collectively enforce a logical sequence for object handling, ensuring that tasks such as arranging or placing objects follow a predefined hierarchical structure that is suggested to LLM along with the task prompt.

4.2. Semantic tagging

The proposed system relies significantly on semantic tagging and recognizing tasks and objects from user input. We use the SpaCy library (https://spacy.io/), which provides advanced capabilities for part-of-speech (POS) tagging and dependency parsing. It takes user input in the form of natural language and extracts the key `tasks` (verbs) and `objects` (nouns) by analyzing the grammatical structure of the sentence.

The user input string is first tokenized into individual words using SpaCy's tokenization. Each token is annotated with a POS tag that identifies its grammatical role, such as a verb, noun, or preposition. The system iterates over the tokens and identifies the verbs using their POS tag. The task is then extracted by selecting verbs that correspond to a predefined set of valid tasks. We assume that the set of tasks that a robot can perform are: `tasks = {''clean'', ''arrange'', ''put'', ''serve'', ''stack''}`.

Formally, the input sentence can be represented as a sequence of tokens, i.e., $\{t_1, t_2, \ldots, t_n\}$. Each token t_i is assigned a part-of-speech tag p_i.

Where $p_i \in \{\text{VERB}, \text{NOUN}, \text{ADJ}, \ldots\}$. The system identifies a task α such that: $\alpha = t_i \quad \text{if} \quad p_i = \text{VERB} \quad \text{and} \quad t_i \in \texttt{tasks}$.

Once the task is identified, the next step is to extract the objects of the task. We use SpaCy's dependency parsing to identify the direct objects (`dobj`), prepositional objects (`pobj`), and conjunct objects (`conj`). These dependencies are used to capture the relationships between the verbs and their corresponding objects. If t_j represents a token with POS tag `NOUN`, then t_j is extracted as an object if it satisfies the condition: $\text{Dependency}(t_j) \in \{\texttt{dobj}, \texttt{pobj}, \texttt{conj}\}$

In addition to processing single-word objects, the system is designed to handle compound nouns connected by underscores (e.g., *green_cup*) or adjectives. In the case of compound nouns, the dependency tag `compound` is used to combine modifiers with the noun. The system ensures that compound words are grouped together, using an underscore (_) to create a single object label (e.g., *green_cup*). Formally, if a noun t_k has a compound modifier t_m, then the object O is given by: $O = \{t_m\_t_k\}$, where $\text{Dependency}(t_m) = \texttt{compound}$.

4.3. Contextual inference module

Once the task and the list of objects are identified, as explained in Section 4.2, the inference module applies a set of SPARQL queries to the ontological knowledge explained in Section 4.1. These queries are designed to generate textual descriptions that facilitate the LLM in computing a semantically correct task plan. The ontology is represented in RDF (Resource Description Framework) format, which provides a standard way of encoding information about objects, actions, and relationships. Using the rdflib library (https://rdflib.readthedocs.io/en/stable/), we load the ontology as RDF graphs, which allows us to perform SPARQL queries to access and extract specific knowledge that is essential for the task-planning process. These queries retrieve information about objects, their classifications, and prioritized actions. By executing these queries, the system formulates natural language prompts and then passes to the LLM alongside user input to ensure that the robot executes actions in a logically and semantically accurate sequence.

The first set of SPARQL queries identifies the types of objects referenced in a user command. This classification is essential, as different objects have specific priorities depending on their type and the task at hand. For example, the query below identifies whether an object, such as an "apple" or "plate", is categorized as a *FoodItem*, *Crockery*, or *Container*.

```
PREFIX ex:
<http://.../kitchen_ontology#>
SELECT ?type WHERE {
    ?obj a ?type ;
    rdfs:label ?label .
    FILTER (lcase(str(?label)) =
        "object_name.lower()")
    FILTER (?type !=
        owl:NamedIndividual) }
```

This query is executed by iterating over items identified in the user's input, enabling the system to classify each item and prepare for a task-specific prioritization. The returned object types inform subsequent queries about priorities and relationships, ensuring that the instructions provided to the LLM are contextually accurate.

Once object types are identified, the inference module executes another set of queries to retrieve predefined task descriptions and priorities from the ontology. These descriptions specify the correct order for performing actions based on the object type and task type (e.g., "put", "clean", or "arrange"). For example, a query to retrieve the priority for any particular action is as follows:

```
PREFIX ex:
http://.../kitchen_ontology#>
SELECT ?priority ?description WHERE {
    ?rule rdf:type ex:ActionPriority ;
     ex:hasAction "action" ;
     ex:hasObjectType "object_type" ;
     ex:hasPriority ?priority ;
     ex:hasDescription ?description . }
```

This query fetches the action priority and a natural language description for each object, allowing the reasoner to compile a comprehensive instruction set. For example, if a *Crockery* item like a "plate" has a higher priority than a *FoodItem* like an "apple" for the "put" action, the system will generate a prompt indicating that the plate should be placed before the apple. This description is adapted to align with the task type, enhancing the LLM's ability to generate a structured and semantically correct plan.

4.4. Perception and env. State descriptor

The perception module and the environment state descriptor are responsible for detecting and identifying objects in the environment and generating a textual description of the scene based on the types of objects and their spatial relationships. The perception module uses a YOLO-based object detection model and NVIDIA's FoundationPose library, for object recognition and precise pose estimation, respectively. The environment state descriptor applies an ontology-based semantic classification, that allows the system to produce structured, detailed descriptions of the environment, which is useful for understanding the environment state. It is important to clarify that the ontology used in the proposed framework is a pre-defined domain ontology and is not dynamically constructed or updated from perception outputs. The ontology is queried only for semantic classification and task-relevant relations, while physical attributes such as object pose and geometry are obtained directly from perception and object models.

The perception module begins with YOLO, which detects objects in the input image. Each detected object is labeled with a class name, such as "apple" or "cup", based on the YOLO that is fine-tuned on YCB dataset, allowing the system to associate each detected object with its specific label. Once objects are detected and labeled, the same image along with point cloud data is processed by the FoundationPose library to compute the precise 3D pose of each object. The environment state descriptor uses the object names and spatial attributes determined by the perception module and applies a set of SPARQL queries on the ontological knowledge to classify each detected object (such as apple and plate as a FoodItem and Crockery) and retrieve relevant geometric information such as object bounding box, object length, and width. The environment state descriptor uses the object names provided by the perception module and applies a set of SPARQL queries on the ontological knowledge to classify each detected object into semantic categories (e.g., apple as a FoodItem and plate as Crockery). Object pose information is obtained from FoundationPose, while geometric attributes such as bounding box, object length, and width are retrieved from the corresponding 3D object models rather than from the ontology.

All extracted information is compiled into a textual description that combines the object's semantic category obtained from the ontology with its spatial and geometric attributes computed from perception and 3D object models. For instance, after processing the object "apple" the system generates a description indicating the "apple" is a "FoodItem" located at position [x,y,z] and orientation [yaw] with a bounding box spanning from [xmin, ymin] to [xmax, ymax] and dimensions of l meters in length and w meters in width. This is done in a similar way as well for all the other detected objects. This textualized environment state description is then fed to the prompt generator that develops the final prompt for the LLM task planner.

4.5. Prompt generator

This module generates the prompt to be sent to the LLM-TAMP module by combining the prompt template with the information coming from the *Contextual Inference Engine* and *Environmental State Descriptor* modules. For instance, in a user input prompt such as "`put the apple and bowl on the tray`", the actual final output prompt feed to the LLM-Task Planner after tuning will be the one shown in Fig. 4 inside the prompt Generated box, where the prompt template is completed with the yellow text that comes from the Contextual Inference Engine, the blue text that comes from the Environment State Descriptor, and the green text that comes from the user input.

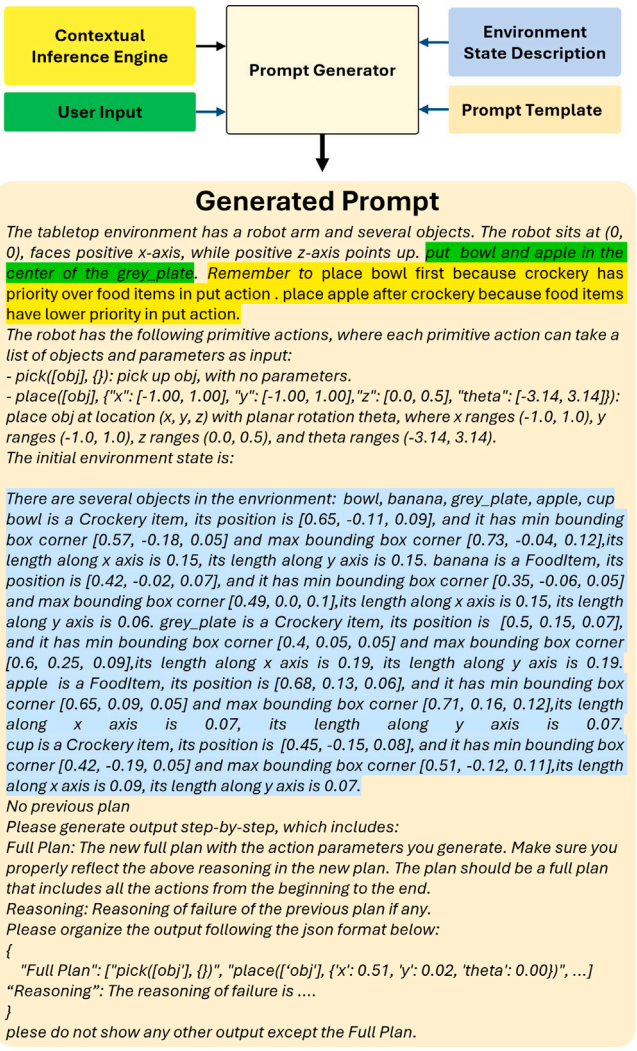

Fig. 4. Illustrate how the prompt generator integrates the information of user input (green), contextual inference module (yellow), Environment State Description (blue), and Prompt Template (black), to construct the final system prompt. The Generated Prompt incorporates structured environment data, action constraints, and reasoning to guide robotic decision-making effectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4.6. LLM task and motion planner

The LLM-Task planner takes a generated prompt as input and formulates a symbolic plan to be executed by the robotic system. The symbolic plan generation closely follows the methodology used in the baseline approach [9]. For instance, using the prompt commented in the previous section, the LLM generates the following symbolic plan:

```
Full Plan =
    Pick ([bowl],{})
    Place([bowl]),{x,y,z,θ}
    Pick ([apple],{})
    Place([apple]),{x,y,z,θ}
```

In this example, the generated symbolic plan outlines a sequential set of actions in which the robot system first picks the plate from its initial position and places it at the specified target location with a given orientation. The process is then repeated for the apple, where the

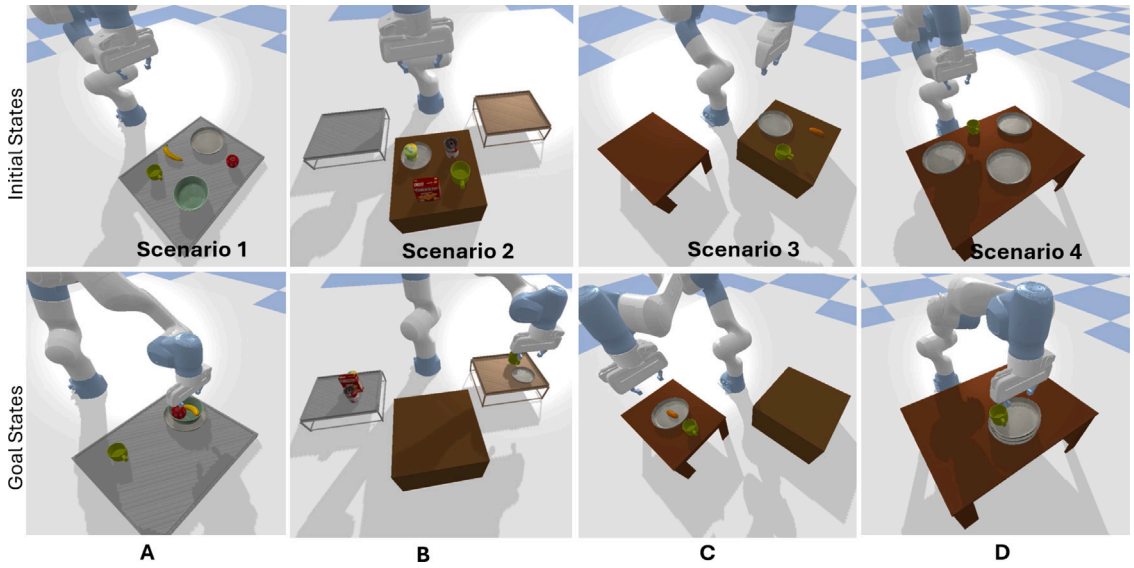

Fig. 5. Example scenarios validating the proposed approach: the first row shows the initial states and the second row shows the goal state. each scenario is used to perform multiple tasks, some of them are given below: (A) Task: *Put bowl, banana, and apple on the plate*; (B) Task: *Clean table, move sugar box, tomato can, and cracker box to the left table, move the plate and cup to the right table*; (C) Task: *Serve breakfast by placing plate, bread, and cup on the table*; (D) Task: *Stack plate1, plate2, and cup on plate3*.

planner generates a `Pick` command followed by a `Place` command, each with their designated positions and orientations.

The execution of each action in this symbolic plan is managed by a motion planning module that interfaces with a motion planner to compute collision-free paths for the manipulator. For motion planning, we employ RRTConnect, which is well-suited for generating efficient and collision-free trajectories in constrained environments. After each action, the motion planner provides feedback to the LLM-Task Planner, confirming successful execution or detailing any encountered issues.

In the case of a motion planning failure, e.g. when a target configuration is unreachable or no collision-free path exists, the motion planner will communicate the specific reason for failure to the LLM-Task Planner. Upon receiving this feedback, the LLM-Task Planner adapts by re-evaluating and recalculating the symbolic plan, taking into account the failure conditions. This iterative feedback mechanism ensures robust task execution by enabling the system to dynamically adjust its planning in response to unexpected obstacles or constraints, optimizing both adaptability and reliability in task completion.

5. Results and discussion

This section provides a comprehensive evaluation of the performance of the Onto-LLM-TAMP framework and its critical components, including semantic tagging accuracy, the effectiveness of knowledge-based reasoning, and comparative benchmarking of various LLM models.

5.1. Baseline: LLM-TAMP

As a comparative baseline, we consider the LLM-TAMP framework [9], a recent Large Language Model-based task and motion planning approach. LLM-TAMP used a pre-trained LLM as a domain-independent planner that jointly proposes symbolic action sequences and continuous action parameters for motion planning. LLM-TAMP incorporates motion planning feedback through iterative prompting, allowing the LLM to refine its proposals by reasoning about motion failures and adjusting subsequent action proposals accordingly. This baseline eliminates the need for manually designed domain-specific interfaces between task and motion planning components and has been demonstrated to effectively solve TAMP problems in simulation and real-world experiments, particularly in structured domains. In our evaluations, the baseline is implemented following the methodology described in [9], ensuring consistency with its original formulation for a fair comparison with the proposed Onto-LLM-TAMP framework.

5.2. Experimental setup

The proposed Onto-LLM-TAMP framework was comprehensively validated through both simulation and real-world experiments, confirming its practical applicability and robustness across diverse task scenarios. The experiments were conducted using a Franka Emika robotic manipulator equipped with its default parallel gripper. The robot provides 7 degrees of freedom and is widely used for manipulation research due to its precision, compliance, and safety features. All manipulation tasks were executed using this standard end-effector configuration without additional tooling.

For perception, an Intel RealSense D450 RGB-D camera was used to capture synchronized color and depth images of the workspace. The camera was mounted to provide a clear view of the manipulation area and supplied input to the perception module for object detection and pose estimation. Object detection was performed using a YOLO-based model, while precise 6D object pose estimation was obtained using NVIDIA's FoundationPose library. The objects used in both simulation and real-world experiments were selected from the YCB object set, which provides standardized object models commonly used for benchmarking robotic manipulation systems. The corresponding 3D object models from the YCB dataset were used to obtain object geometry information, including bounding box dimensions, which supported motion planning and environment state description.

Fig. 5 illustrates the simulation scenarios used for validation. These scenarios were implemented using the PyBullet physics engine. Four different task scenarios were designed to thoroughly evaluate the system's performance.

In the first scenario (Fig. 5-A), the task objective involved organizing items by placing an `apple`, a `banana`, and a `bowl` onto a `plate`. The second scenario (Fig. 5-B) focused on a table-cleaning task that required the relocation of various objects; specifically, crockery items (`cup` and `plate`) had to be moved to the `left_table`, whereas boxed food items (`cracker_box`, `sugar_box`, and `tomato_can`)

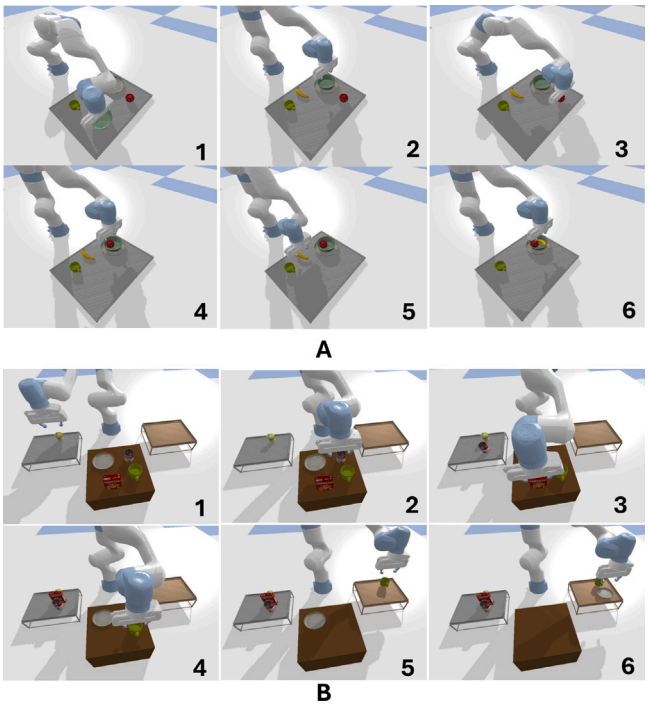

Fig. 6. Sequence of snapshots of the following tasks: (A) Task: *Put apple, banana, and bowl in plate*; (B) Task: *Clean table, move plate and cup to the right_table, move sugar_box, tomato_can, and cracker_box to the left_table.*.

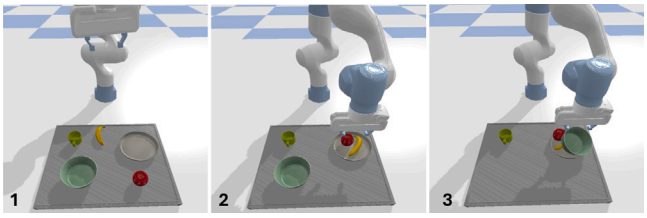

Fig. 7. Example of a wrong symbolic plan computation. for the Task: *Put apple, banana, and bowl in plate*.

were to be placed on the `right_table`. The sequence of execution is depicted in Fig. 6.

The third scenario (Fig. 5-C) simulated arranging a breakfast table, involves the precise placement of a `cup` and `plate` on the table, followed by putting a `bread` on the plate. The final scenario (Fig. 5-D) is about stacking objects, where (`plate1`, `plate2`, and `cup`) had to be systematically stacked onto `plate3`. To further assess the system's robustness, variations in task prompts were introduced by changing the order of objects mentioned by the user. These scenarios were carefully designed to evaluate the system's capability in managing complex arrangements of objects, where maintaining the correct temporal sequence of subtasks is critical for achieving semantically correct and successful task execution.

5.3. Onto-LLM-TAMP evaluation

The above scenarios are used to compare the proposed framework with the baseline approach. For statistical reliability, each experiment was repeated 100 times for every task, and all reported metrics correspond to the average performance across these runs. The results presented in Table 1 highlight the importance of integrating ontology-based reasoning into LLM-TAMP systems. The parameters we compare are;

- *Task planning success rate (TPSR)*: Represents the percentage of tasks for which the generated plans by the LLM correctly capture the intended task semantics and objectives, resulting in an executable action sequence.
- *Execution success rate (EXESR)*: Indicates the percentage of tasks successfully completed when executing the generated plans on the robot, highlighting the practical feasibility and effectiveness of the plans.
- *Number of LLM calls (# CALLs)*: Measures how many LLM calls were required to generate a suitable action plan, reflecting the efficiency and complexity involved in the planning process.

The Onto-LLM-TAMP consistently outperforms the baseline LLM-TAMP, particularly in complex scenarios that require refined reasoning and object categorization. For straightforward tasks with explicit and correctly ordered object descriptions, such as *Task 1: Put bowl, banana, and apple in plate*, both Onto-LLM-TAMP and traditional LLM-TAMP systems achieve nearly perfect performance (approximately 99.5% TPSR and above 97% EXESR), requiring only a minimal number of LLM calls. However, when the order of objects is altered, as seen in *Task 2*, the traditional LLM-TAMP's performance significantly drops to about 30.7% TPSR and 29.4% EXESR, along with an increased number of LLM calls (8.6 on average). In contrast, Onto-LLM-TAMP maintains high performance (99.4% TPSR and 96.9% EXESR) with fewer LLM calls (2.6 on average). This robustness improvement for the Onto-LLM-TAMP is due to the explicit incorporation of the correct semantic sequence of actions using knowledge-based reasoning, which reduces ambiguity in the task. The reason for the failure of LLM-TAMP is that in most of the cases, it ended up performing the task as shown in Fig. 7.

In more complex scenarios, such as *Task 3*, which involves detailed instructions about moving various items (crockery and boxed foods) to specific locations, both approaches initially perform similarly with high success rates (over 96% TPSR and around 90% EXESR). However, when the task description sequence is reversed and made ambiguous (as in *Task 4* and *Task 5*), the baseline approach suffers a significant performance decline, reaching as low as 0% in TPSR and EXESR. The Onto-LLM-TAMP, meanwhile, maintains a consistently high-performance level (TPSR around 98% and EXESR around 91% for *Task 4*, and modest success of approximately 48% TPSR for the most ambiguous *Task 5*). This capability results from using structured ontological knowledge to precisely categorize items (e.g., identifying `cracker_box` as boxed food and `plate` as crockery), thus providing the LLM with clearer contextual information.

For tasks that involved serving and stacking items (*Tasks* 6 to 10), the Onto-LLM-TAMP consistently demonstrates superior performance compared to the baseline. In *Tasks 6 and 7*, which involve serving breakfast, the baseline's TPSR drops to around 20% when the item order is changed. In contrast, the Onto-LLM-TAMP maintains consistently high success rates (above 98% TPSR and around 92% EXESR). Similarly, in stacking tasks (*Tasks 8 and 9*), the Onto-LLM-TAMP achieves high TPSR (around 98%) and EXESR (above 92%), significantly outperforming the baseline, especially when the item order is varied. *Task 10*, involving stacking items ambiguously described as *cup and plates on the table*, highlights the limits of both approaches; the Onto-LLM-TAMP approach still achieves success (68.1% TPSR and 61.8% EXESR) because the knowledge reasoner described: *the objects with the large bounding boxes will come first in stacking*. These results indicate the Onto-LLM-TAMP approach's capability to handle ambiguity and complex semantic scenarios more effectively.

During the real-world demonstration, we execute the task of *put a banana, an apple, and a bowl in the plate*, showcasing the system's ability to accurately detect, localize, and manipulate objects in a real environment. Fig. 8 presents a series of screenshots capturing key moments of the execution process. To ensure precise object identification and spatial positioning, we utilize YOLO-V8 for object detection and FoundationPose for pose estimation. YOLO-V8 enables real-time recognition

Table 1
Comparison of LLM-TAMP and Onto-LLM-TAMP. TPSR: Task planning success rate, EXESR: Execution success rate, CALLs: LLM Calls.

Prompt	LLM-TAMP			Onto-LLM-TAMP		
	TPSR %	EXESR %	# CALLs	TPSR #	EXESR #	# CALLs
Task 1: Put bowl, banana and apple in plate	99.5	97.3	2.8	99.7	97.1	3.2
Task 2: Put banana, apple and bowl in plate	30.7	29.4	8.6	99.4	96.9	2.6
Task 3: Clean table, move sugar_box, tomato_can, and cracker_box to the left table, move plate and cup to the right table	96.4	89.3	7.7	98.8	93.2	6.3
Task 4: Clean table, move plate and cup to the right table, move sugar_box, tomato_can, and cracker_box to the left table	42.8	36.6	8.4	98.3	91.4	6.8
Task 5: Clean table, move crockery items to the right table, move boxed food items to the left table	0	0	10	48.1	41.6	8.3
Task 6: Serve breakfast by placing plate, bread and cup on the table.	98.4	90.3	4.6	98.7	90.1	4.9
Task 7: Serve breakfast by placing bread, plate and cup on the table.	20.8	19.6	9.7	98.6	92.4	4.3
Task 8: Stack plate1, plate2 and cup on plate3	98.9	94.8	6.7	98.7	92.1	5.3
Task 9: Stack cup, plate1, and plate2 on plate3	17.2	12.4	9.8	98.1	93.2	5.8
Task 10: Stack cup and plates on the table	0	0	10	68.1	61.8	6.3

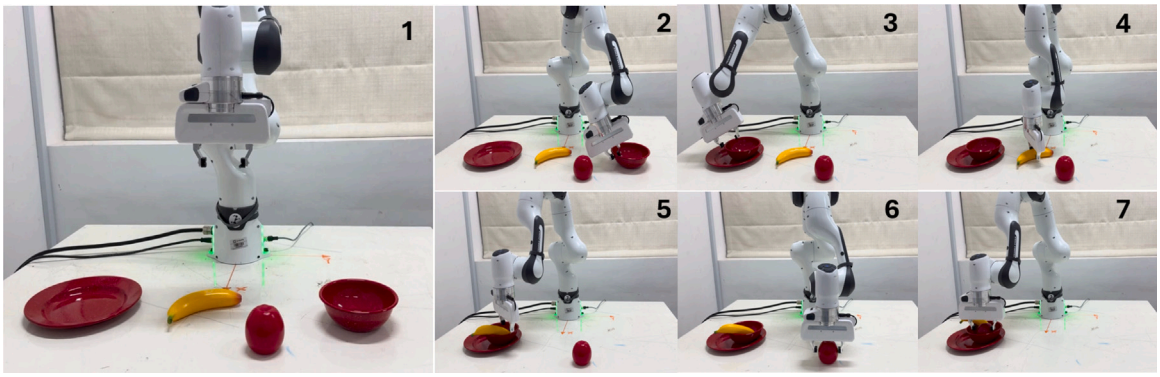

Fig. 8. Screenshots of the execution in the real environment. The task was to *put banana, apple and bowl into the plate*.

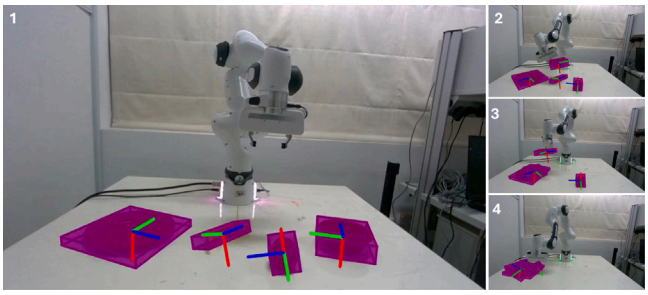

Fig. 9. Screenshots of pose estimation during the execution in the real environment. The task was to put banana, apple, and bowl into the plate.

of objects in the scene, while FoundationPose estimates their positions and orientations, allowing the robot to plan and execute accurate pick-and-place actions. Figs. 9 and 10 illustrate the intermediate steps, including detected objects and their estimated poses, demonstrating the effectiveness of our approach in real-world conditions.

5.4. Semantic tagging and reasoning evaluation

Semantic tagging and knowledge-based reasoning are two core components of the proposed framework and operate as *sequential stages within a single processing pipeline*. Semantic tagging first analyzes the input prompt to identify the intended actions and associated objects. The outputs of this stage are then passed to the knowledge-based reasoner, which utilizes the extracted actions and objects to enrich and clarify the input by incorporating ontological knowledge, resulting in a more precise and semantically consistent task description.

Table 2 reports the computational time and accuracy of each stage. The objective of this evaluation is not to compare the performance of semantic tagging and reasoning against each other, but rather to analyze their individual behavior and contribution within the overall prompt-tuning process.

- *Time:* This metric quantifies the processing time required by each stage in the pipeline. Semantic tagging time reflects the cost of extracting actions and objects from the input prompt, while reasoning time reflects the cost of ontology querying and inference applied to the extracted elements.
- *Accuracy:* These metrics are reported on a per-stage basis. For semantic tagging, accuracy indicates the correctness of identifying relevant actions and objects from the input prompt. For reasoning, accuracy reflects how reliably the ontology-based inference enhances the prompt by applying correct semantic relationships and task constraints, given the outputs of the tagging stage.

The reported metrics highlight the complementary roles of the two stages. Semantic tagging is consistently lightweight and fast, while the reasoning stage introduces additional computational overhead due to ontology-driven inference. This overhead is intentional and necessary

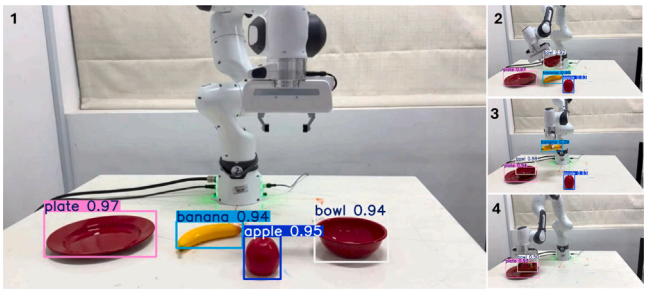

Fig. 10. Screenshots of object identification during the execution in the real environment. The task was to put banana, apple, and bowl into the plate.

Table 2
Processing time and accuracy of semantic tagging (ST) and reasoning process (RP) as sequential stages of the prompt-tuning pipeline. The metrics are reported to characterize the contribution of each stage and are not intended as a direct performance comparison.

Experiment	Time (s)		Accuracy (%)	
	ST	RP	ST	RP
Exp 1 (Task 1-10)	0.0048	0.094	98.43	89.71
Exp 2 (Random Tasks)	0.0071	0.109	81.51	77.94

to enforce semantic constraints such as task prioritization and temporal ordering.

In Experiment 1, semantic tagging and reasoning were applied to each task ten times, and average processing time and accuracy were computed across ten structured tasks. Semantic tagging requires an average of 0.0048 s with an accuracy of 98.43%, indicating that for well-structured prompts the tagging stage efficiently and reliably extracts relevant actions and objects. Given these extracted entities, the reasoning stage reports an average processing time of 0.094 s with an accuracy of 89.71%, showing that ontology-based inference is generally effective in enriching the prompt with semantically meaningful constraints. Occasional failures are mainly due to incomplete or ambiguous entity definitions rather than reasoning errors.

In Experiment 2, random task prompts were used by introducing typos, misplaced punctuation, and variations in linguistic structure. Under these conditions, semantic tagging time increases slightly to 0.0071 s, reflecting the additional effort required to process less predictable language. The corresponding accuracy of 81.51% indicates that most actions and objects are still correctly identified, although ambiguity and noise in the input reduce extraction reliability. The reasoning stage requires an average of 0.109 s, reflecting the increased complexity of ontology matching under noisy inputs. Its accuracy of 77.94% shows that ontology-based inference remains effective in most cases, while errors primarily stem from imperfect upstream tagging or unfamiliar linguistic formulations.

Overall, the results in this section demonstrate that semantic tagging and ontology-based reasoning together form an efficient and reliable prompt-tuning pipeline. Semantic tagging provides fast and accurate extraction of task-relevant entities under structured inputs, while ontology-based reasoning introduces a modest but necessary computational overhead to enforce semantic consistency and task-level constraints. Importantly, the observed performance characteristics should be interpreted at the pipeline level rather than as a direct comparison between the two stages. Despite increased linguistic variability in unstructured prompts, the combined pipeline remains robust and incurs negligible overhead relative to the overall LLM inference time, while significantly improving the semantic correctness of generated task plans.

5.5. LLM models evaluations

The quality and correctness of the computed symbolic plan largely depend on the clarity of the prompt and the capabilities of the LLM model. To evaluate this, we assessed the performance of various LLM models, specifically GPT, LLaMA, Gemini, and Cohere in both standard LLM-TAMP and onto-LLM-TAMP frameworks. The results of this evaluation are summarized in Table 3. The experiments measure two key parameters for both approaches:

- *Time:* Defined as the duration each LLM requires to generate executable task plans using API call.
- *Correctness:*, Assessed by two robotics experts who manually evaluated each generated action sequence. The evaluation criteria combined semantic accuracy (90%) and structural correctness (10%). High correctness scores indicate that the generated plans are not only syntactically executable but also semantically coherent and realistically applicable to robotic scenarios.

For tasks with straightforward, clearly structured descriptions (e.g., Task 1 and Task 8), both methods performed comparably well, achieving high correctness scores (ranging from approximately 92% to 100%). This consistency demonstrates that, when task instructions are explicit and sequentially correct, LLM-based approaches reliably translate user prompts into executable task plans. Time performance in such tasks varied slightly while generally remaining stable across models and methods. However, when tasks contained ambiguity, non-explicit ordering, or object arrangements different from the correct execution sequence (e.g., Task 2: “Put banana, apple, and bowl in plate”, Task 4, and Task 5), the performance of the LLM models significantly declined. It can be observed by correctness scores that drop sharply (often to 0%–10%) when the user task prompt is directly provided as input to the LLM model (LLM-TAMP). Such tasks posed a major challenge for LLM-TAMP due to the inherent ambiguity or lack of structured context in the provided input, leading to an unreliable sequence of actions. In contrast, when the user prompts along with the ontological reasoning passed to the LLM Models (onto-LLM-TAMP), it consistently maintained high correctness (usually around 100%), clearly showcasing its ability to handle ambiguity through ontological knowledge integration effectively. By utilizing structured domain knowledge of onto-LLM-TAMP, the LLM models were able to interpret ambiguous instructions more precisely, ensuring semantically coherent action sequences.

Interestingly, even though onto-LLM-TAMP involves an additional reasoning step that potentially increases the overall processing complexity, it still achieves comparable execution times in most of the cases. This indicates that ontology-driven clarification of task semantics can streamline the ensuing LLM processing by providing clearer input (see Fig. 11).

The average results presented at the end of Table 3 clearly illustrate the overall advantages of employing an Onto-LLM-TAMP approach compared to the standard LLM-TAMP setting across all evaluated models. Specifically, the average correctness improved significantly when using the ontological enhancement, reflecting enhanced semantic understanding and reduced ambiguity in generated action sequences. For clarity, we have added a histogram showing the average correctness score of LLM models for the cases of LLM-TAMP and Onto-LLM-TAMP. In correctness, GPT-4 showed an improvement from 55% correctness in standard LLM-TAMP to 91% correctness with ontology integration. Similar trends are observed for Gemini (from 55% to 83%), Cohere (from 45% to 63%), and LLaMA (from 34% to 74%).

In terms of computational efficiency, measured by the average API calls response time, the Onto-LLM-TAMP maintained comparable response times compared to the basic LLM-TAMP method. Fig. 12 shows the histogram of the comparison of average API call time of LLM models in LLM-TAMP and Onto-LLM-TAMP. For GPT-4 and LLaMA, the

Table 3
"Comparison of LLM models on Tasks 1 to 10 for LLM-TAMP and Onto-LLM-TAMP, evaluated based on two key metrics: execution time (T) in seconds and correctness (C) score in %.

Task	GPT-4				Gemini				Cohere				LLaMA			
	LLM-TAMP		Onto-LLM-TAMP		LLM-TAMP		Onto-LLM-TAMP		LLM-TAMP		Onto-LLM-TAMP		LLM-TAMP		Onto-LLM-TAMP	
	T	C	T	C	T	C	T	C	T	C	T	C	T	C	T	C
1	2.55	100	2.62	100	2.31	100	2.15	100	3.01	100	3.03	100	3.92	100	3.98	100
2	3.25	0	3.61	100	2.10	0	2.41	100	3.02	10	2.44	100	3.57	0	3.32	100
3	3.19	100	4.87	100	3.47	100	2.96	100	3.07	30	3.35	30	4.91	90	5.28	90
4	2.86	10	3.66	100	3.53	10	3.48	100	3.06	0	3.49	0	3.75	10	2.89	100
5	3.12	10	3.80	100	3.53	10	3.48	100	3.61	0	2.90	0	4.61	10	4.96	50
6	3.23	100	3.72	100	1.99	100	2.71	100	3.75	90	3.24	90	4.75	90	4.45	90
7	2.60	10	3.70	100	2.68	10	2.74	100	4.23	10	3.31	90	5.09	0	4.95	90
8	3.60	100	3.62	100	2.93	100	2.06	10	3.18	100	3.92	100	3.22	10	4.91	10
9	3.82	10	3.72	100	2.50	10	2.86	10	4.01	10	3.28	10	4.03	10	4.23	10
10	4.29	10	4.65	10	2.86	10	2.74	10	3.35	10	3.12	10	4.04	10	4.80	100
Average	**3.25**	**55.0**	**3.80**	**91.0**	**2.79**	**55.0**	**2.76**	**83.0**	**3.43**	**36.0**	**3.21**	**63.0**	**4.19**	**34.0**	**4.28**	**74.0**

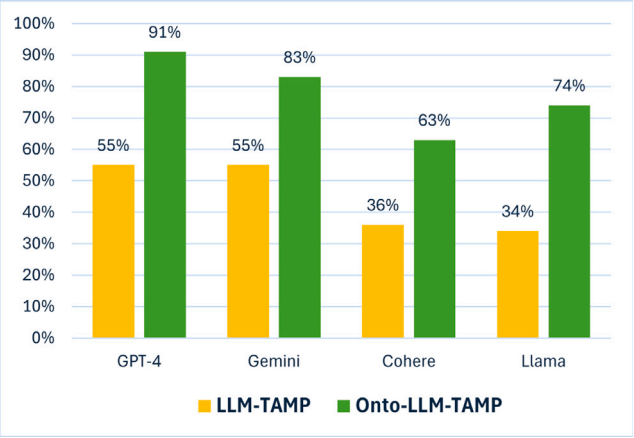

Fig. 11. Histogram showing the average correctness scores of LLM models for LLM-TAMP and Onto-LLM-TAMP cases.

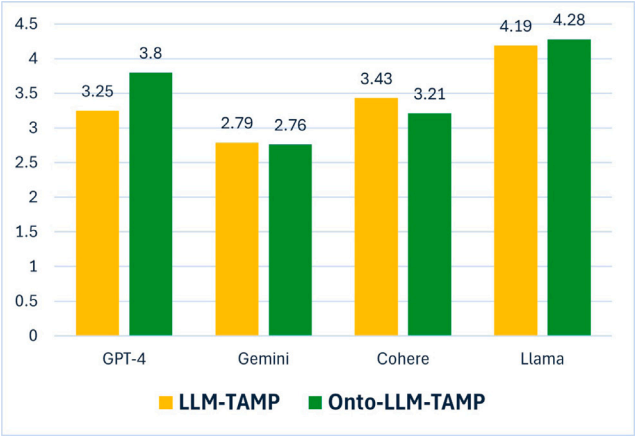

Fig. 12. Histogram showing the average API call time of LLM models for LLM-TAMP and Onto-LLM-TAMP cases.

Onto-LLM-TAMP required a modest increase in average time (3.25s to 3.80s and 4.19s to 4.28s, respectively). Conversely, Gemini and Cohere remained stable (Gemini: 2.79s to 2.76s; Cohere: 3.43s to 3.21s). These variations indicate that while ontology integration slightly affects response times, the substantial improvements in correctness significantly surpass minor time trade-offs.

5.6. Motion planning time evaluation

Motion planning is another critical component of the proposed framework. Its performance significantly influences the overall efficiency of task execution. To evaluate this aspect, we compared the motion planning performance of the standard LLM-TAMP and the Onto-LLM-TAMP approaches using the RRTConnect algorithm. The results are summarized in Table 4. Motion planning times were calculated as the average planning time of multiple calls of the motion planning during the execution of the subtask of each given task.

For relatively simple tasks (Tasks 1, 3, 6, and 8), both approaches exhibited comparable planning times, indicating the minimal impact of ontological reasoning when task instructions are clear and straightforward. For instance, Task 1 demonstrated similar results (0.13s vs. 0.11s), while slightly more complex tasks, like Task 3, showed nearly identical performance (0.61s vs. 0.60s). However, even moderately complex tasks with slight ambiguity, such as Task 2 and Task 7, significantly benefited from ontology-driven reasoning, resulting in

Table 4
Average Motion Planning Time (seconds) for Tasks 1–10, Comparing LLM-TAMP and Onto-LLM-TAMP.

Task	Motion planning time (s)	
	LLM-TAMP	Onto-LLM-TAMP
Task 1	0.13	0.11
Task 2	2.62	0.14
Task 3	0.61	0.60
Task 4	6.48	0.93
Task 5	NA	3.72
Task 6	0.33	0.28
Task 7	1.89	0.30
Task 8	0.27	0.25
Task 9	4.36	0.23
Task 10	NA	1.59

notably reduced planning times (from 2.62s to 0.14s and 1.89s to 0.30s, respectively). Ontology-driven reasoning showed significant advantages in complex or ambiguous scenarios (Tasks 4, 5, 9, and 10). For example, in the case of Task 4, planning times were reduced from 6.48s to 0.93s. Tasks 5 and 10, completely failed under standard LLM-TAMP, and succeeded efficiently (3.72 and 1.59s, respectively) with the Onto-LLM-TAMP.

The primary reason for the significant reduction in planning time observed with the Onto-LLM-TAMP is that, in scenarios where task

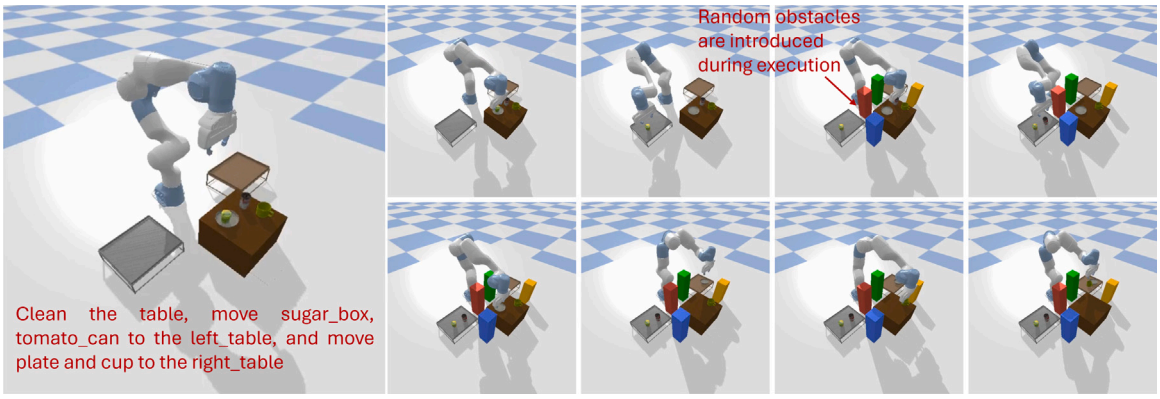

Fig. 13. Dynamic obstacle experiment. Random obstacles are introduced during task execution, requiring online replanning and adaptation. The robot successfully completes the task by updating environment state descriptions and maintaining semantic task consistency.

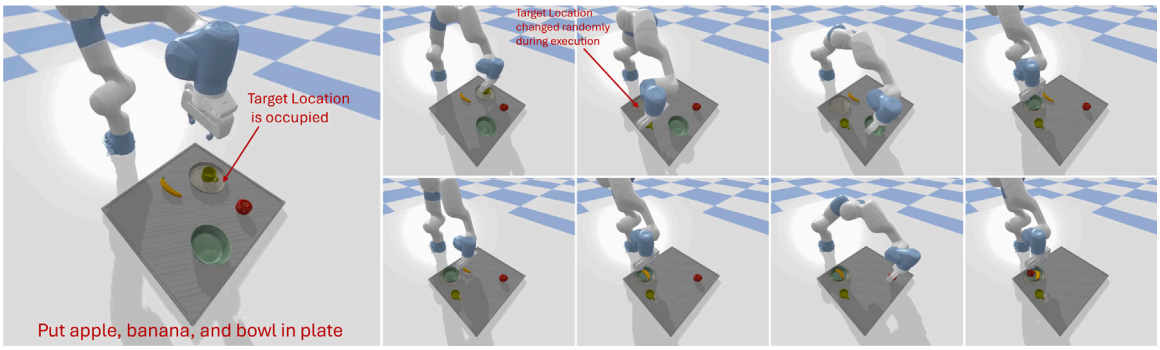

Fig. 14. Dynamic target experiment. The target placement location is initially occupied and later moved during execution. The proposed framework adapts to both conditions by reasoning over semantic constraints and updated environment states.

instructions or object ordering are ambiguous or incorrect, the conventional LLM-TAMP method often struggles to find collision-free trajectories. This struggle causes repeated failures at the motion planning stage, which then prompts the LLM planner to recompute the symbolic plan continuously, consequently increasing the total motion planning duration, particularly in complex tasks.

5.7. Evaluation in dynamic environments

To further evaluate the generalizability and adaptability of the proposed Onto-LLM-TAMP framework, we introduce two additional experiments that explicitly incorporate dynamic environmental changes and more complex planning structures.

5.7.1. Dynamic obstacle handling

In this experiment, the robot performs table-cleaning and object relocation tasks while random obstacles are introduced during execution (Fig. 13). These obstacles appear at arbitrary positions in the workspace after the task has started, potentially blocking previously feasible motion paths. As a result, the system must continuously update its environment state description and adapt both symbolic and geometric plans to ensure collision-free execution.

The proposed framework successfully detects the updated environment state, regenerates task-consistent plans, and completes the task without manual intervention. This experiment demonstrates the system's ability to handle dynamic spatial constraints that arise during execution, rather than being predefined at planning time.

5.7.2. Dynamic target accessibility and repositioning

The second experiment focuses on a hierarchical placement task in which the target container (plate) is initially occupied by another object, rendering the target location unavailable. During execution, the target plate is also relocated to a new position (Fig. 14). The robot must therefore (i) reason that the target must be cleared before placement and (ii) adapt to the updated target location.

This scenario introduces both semantic and temporal dependencies, as clearing the target is a prerequisite for task completion and the target pose is not fixed throughout execution. The Onto-LLM-TAMP framework successfully handles both conditions by updating the environment state description and enforcing ontological task constraints, leading to correct task execution despite dynamic goal changes.

Together, these experiments demonstrate that the proposed framework is not limited to static environments or instruction-level variations. Instead, it can robustly adapt to dynamic environmental changes, including unexpected obstacle and moving targets.

5.8. Discussion and limitations

Semantic tagging plays a central role in the proposed Onto-LLM-TAMP framework, as it provides the structured representation of actions and objects required for ontology-based reasoning and prompt elaboration. In the current implementation, extracted entities are aligned with predefined ontology concepts, enabling consistent semantic classification, task-level prioritization, and temporal ordering. This design choice supports reliable reasoning and interpretable task planning in structured domains, which is particularly important for maintaining semantic consistency during execution.

At the same time, this reliance on predefined ontology concepts introduces a design assumption that the task vocabulary and object categories are known a priori. While this assumption is reasonable for many controlled environments and application-specific domains, it may limit flexibility when extending the framework to more diverse or open-ended task descriptions involving previously unseen objects or concepts. This reflects a common trade-off in ontology-guided planning between semantic structure and generality.

An important direction for future work is to relax this assumption by enabling more adaptive ontology management. One promising research direction is to dynamically extend the ontology using Large Language Models in conjunction with environment state descriptors, allowing new object categories and semantic relations to be incorporated at runtime when previously unknown objects are encountered. Such an approach could preserve the benefits of ontology-guided task-level reasoning while improving adaptability and scalability to evolving task domains and open-world environments.

6. Conclusions

This work introduced a novel Onto-LLM-TAMP framework to enhance task and motion planning with LLMs. Integrating knowledge-based reasoning and ontology-driven environment state descriptions, the framework dynamically refines user prompts to generate semantically accurate and context-aware symbolic plans. Unlike static, template-based approaches, it addresses flaws such as incorrect temporal goal ordering and adapts effectively to dynamic environments. Validation in both simulation and real-world scenarios demonstrated significant improvements in planning accuracy and adaptability, highlighting the potential of combining LLMs with ontological reasoning for advanced robotic planning in complex tasks and dynamic contexts.

CRediT authorship contribution statement

Muhayy Ud Din: Writing – review & editing, Writing – original draft, Software, Methodology, Conceptualization. **Jan Rosell:** Supervision, Funding acquisition, Conceptualization. **Waseem Akram:** Writing – original draft, Validation. **Isiah Zaplana:** Methodology, Conceptualization. **Maximo A. Roa:** Writing – review & editing, Methodology, Conceptualization. **Irfan Hussain:** Writing – review & editing, Methodology, Funding acquisition, Conceptualization.

Declaration

During the preparation of this work, the author (s) used ChatGPT and Gemini to improve language and readability. After using this tool/service, the author(s) reviewed and edited the content as needed and takes (s) full responsibility for the content of the publication.

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.

Acknowledgments

This work has been supported by the Khalifa University of Science and Technology under Award No. RC1-2018-KUCARS, Silal Innovation Oasis through the projects under grants 8475000023, and by the European Commission's Horizon Europe Framework Programme with the project IntelliMan (AI-Powered Manipulation System for Advanced Robotic Service, Manufacturing and Prosthetics) under Grant Agreement 101070136.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.robot.2026.105404.

Data availability

No data was used for the research described in the article.

References

[1] C. Aeronautiques, A. Howe, C. Knoblock, I.D. McDermott, A. Ram, M. Veloso, D. Weld, D.W. Sri, A. Barrett, D. Christianson, et al., PDDL the planning domain definition language, Tech. Rep. (1998).
[2] A.L. Blum, M.L. Furst, Fast planning through planning graph analysis, Artificial Intelligence 90 (1–2) (1997) 281–300.
[3] J. Hoffmann, B. Nebel, The FF planning system: Fast plan generation through heuristic search, J. Artificial Intelligence Res. 14 (2001) 253–302.
[4] Y. Chen, J. Arkin, C. Dawson, Y. Zhang, N. Roy, C. Fan, AutoTAMP: Autoregressive task and motion planning with LLMs as translators and checkers, 2023, arXiv Preprint 2306.06531.
[5] Y. Ding, X. Zhang, S. Amiri, N. Cao, H. Yang, A. Kaminski, C. Esselink, S. Zhang, Integrating action knowledge and LLMs for task planning and situation handling in open worlds, Auton. Robots 47 (8) (2023) 981–997.
[6] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F.L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al., GPT-4, technical report, 2023, arXiv Preprint 2303.08774.
[7] 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: Int. Conf. on Robotics and Automation (ICRA), IEEE, 2023, pp. 11523–11530.
[8] Y. Ding, X. Zhang, C. Paxton, S. Zhang, Task and motion planning with large language models for object rearrangement, in: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, IROS, 2023, pp. 2086–2092.
[9] S. Wang, M. Han, Z. Jiao, Z. Zhang, Y.N. Wu, S.-C. Zhu, H. Liu, LLM3: Large language model-based task and motion planning with motion failure reasoning, 2024, arXiv preprint arXiv:2403.11552.
[10] Q. Dong, L. Li, D. Dai, C. Zheng, J. Ma, R. Li, H. Xia, J. Xu, Z. Wu, T. Liu, et al., A survey on in-context learning, 2022, arXiv Preprint 2301.00234.
[11] B. Li, P. Wu, P. Abbeel, J. Malik, Interactive task planning with language models, 2023, arXiv preprint arXiv:2310.10645.
[12] Y. Jin, D. Li, A.V. Yong, J. Shi, P. Hao, F. Sun, J. Zhang, B. Fang, RobotGPT: Robot manipulation learning from ChatGPT, IEEE Robot. Autom. Lett. 9 (2024) 2543–2550.
[13] X. Li, G. Tian, Y. Cui, Fine-grained task planning for service robots based on object ontology knowledge via large language models, IEEE Robot. Autom. Lett. 9 (2024) 6872–6879.
[14] A. Orthey, C. Chamzas, L.E. Kavraki, Sampling-based motion planning: A comparative review, Annu. Rev. Control. Robot. Auton. Syst. 7 (2023).
[15] J. Rosell, R. Suárez, N. García, M.U. Din, Planning grasping motions for humanoid robots, Int. J. Humanoid Robot. 16 (06) (2019) 1950041.
[16] Y.-q. Jiang, S.-q. Zhang, P. Khandelwal, P. Stone, Task planning in robotics: an empirical comparison of pddl-and asp-based systems, Front. Inf. Technol. Electron. Eng. 20 (3) (2019) 363–373.
[17] C.R. Garrett, T. Lozano-Pérez, L.P. Kaelbling, PDDLstream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning, in: Int. Conf. on Automated Planning and Scheduling, vol. 30, 2020, pp. 440–448.
[18] M. Noseworthy, C. Moses, I. Brand, S. Castro, L. Kaelbling, T. Lozano-Pérez, N. Roy, Active learning of abstract plan feasibility, 2021, arXiv Preprint 2107.00683.
[19] Z. Yang, C.R. Garrett, T. Lozano-Pérez, L. Kaelbling, D. Fox, Sequence-based plan feasibility prediction for efficient task and motion planning, 2022, arXiv Preprint 2211.01576.
[20] A. Aliakbar, Muhayyuddin, J. Rosell, Task planning using physics-based heuristics on manipulation actions, in: IEEE Int. Conf. on Emerging Technologies and Factory Automation, ETFA, 2016, pp. 1–8.
[21] A. Akbari, M. Gillani, J. Rosell, Reasoning-based evaluation of manipulation actions for efficient task planning, in: Second Iberian Robotics Conference: Advances in Robotics, Springer, 2015, pp. 69–80.
[22] OpenAI, GPT-4, Technical Report, 2023, arXiv preprint arXiv:2303.08774.
[23] R. Anil, M. Bosma, et al., Gemini: A family of highly capable multimodal models, 2023, arXiv preprint arXiv:2312.11802.
[24] H. Touvron, L. Martin, K. Stone, et al., Llama: Open and efficient foundation language models, 2023, arXiv preprint arXiv:2302.13971.
[25] C. Jiang, Y. Zeng, D.-Y. Yeung, CoherenDream: Boosting holistic text coherence in 3D generation via multimodal large language models feedback, 2025, arXiv preprint arXiv:2504.19860.
[26] Q. Zhu, D. Guo, Z. Shao, D. Yang, P. Wang, R. Xu, Y. Wu, Y. Li, H. Gao, S. Ma, et al., Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence, 2024, arXiv preprint arXiv:2406.11931.
[27] W. Huang, P. Abbeel, D. Pathak, I. Mordatch, Language models as zero-shot planners: Extracting actionable knowledge for embodied agents, in: Int. Conf. on Machine Learning, PMLR, 2022, pp. 9118–9147.
[28] T. Brown, B. Mann, N. Ryder, M. Subbiah, J.D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, Adv. Neural Inf. Process. Syst. 33 (2020) 1877–1901.
[29] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, et al., Do as I can, not as I say: Grounding language in robotic affordances, 2022, arXiv Preprint 2204.01691.

[30] W. Huang, F. Xia, D. Shah, D. Driess, A. Zeng, Y. Lu, P. Florence, I. Mordatch, S. Levine, K. Hausman, et al., Grounded decoding: Guiding text generation with grounded models for robot control, 2023, arXiv Preprint 2303.00855.
[31] S. Vemprala, R. Bonatti, A. Bucker, A. Kapoor, ChatGPT for robotics: Design principles and model abilities, IEEE Access (2023).
[32] W. Huang, et al., Language models as zero-shot planners: Extracting actionable knowledge for embodied agents, 2022, arXiv preprint arXiv:2201.07207.
[33] G. Jocher, J. Qiu, A. Chaurasia, Ultralytics YOLO, 2023.
[34] B. Wen, W. Yang, J. Kautz, S. Birchfield, FoundationPose: Unified 6D pose estimation and tracking of novel objects, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 17868–17879.
[35] O. Ruiz-Celada, A. Dalmases, I. Zaplana, J. Rosell, Smart perception for situation awareness in robotic manipulation tasks, IEEE Access 12 (2024) 53974–53985.