Scalable Task Planning via Large Language Models and Structured World Representations

Pérez-Dattari, Rodrigo; Li, Z.; Babuska, R.; Kober, J.; Della Santina, C.

DOI
10.1002/adrr.202500002

Publication date
2025

Document Version
Final published version

Published in
Advanced Robotics Research

Citation (APA)
Pérez-Dattari, R., Li, Z., Babuska, R., Kober, J., & Della Santina, C. (2025). Scalable Task Planning via Large Language Models and Structured World Representations. *Advanced Robotics Research*, Article e202500002. https://doi.org/10.1002/adrr.202500002

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Advanced Robotics Research

WILEY VCH

ADVANCED ROBOTICS RESEARCH Open Access

RESEARCH ARTICLE OPEN ACCESS

Scalable Task Planning via Large Language Models and Structured World Representations

Rodrigo Pérez-Dattari[1] | Zhaoting Li[1] | Robert Babuška[1,2] | Jens Kober[1] | Cosimo Della Santina[1,3]

[1]Department of Cognitive Robotics, Delft University of Technology, Delft, Netherlands | [2]Czech Institute of Informatics, Robotics, and Cybernetics, Czech Technical University in Prague, Prague, Czech Republic | [3]Institute of Robotics and Mechatronics, German Aerospace Center (DLR), Wessling, Germany

Correspondence: Rodrigo Pérez-Dattari (rodrigoperezd@gmail.com)

Received: 7 January 2025 | **Revised:** 17 September 2025 | **Accepted:** 26 September 2025

Funding: National Growth Fund program NXTGEN Hightech.

Keywords: graphs | LLMs | robotics | task planning | taxonomy

ABSTRACT

Planning methods often struggle with computational intractability when solving task-level problems in large-scale environments. This work explores how the commonsense knowledge encoded in Large Language Models (LLMs) can be leveraged to enhance planning techniques for such complex scenarios. Specifically, we propose an approach that uses LLMs to efficiently prune irrelevant components from the planning problem's state space, thereby substantially reducing its complexity. We demonstrate the efficacy of our system through extensive experiments in a household simulation environment as well as real-world validation on a 7-DoF manipulator (video: https://youtu.be/6ro2UOtOQS4).

1 | Introduction

In homes, hospitals, and factories, there is an increasing demand for robotic assistants capable of responding to high-level commands such as '*Clean room A*.' Search-based planning [1–3] provides a promising framework for generating the action sequences required to solve such tasks. However, these techniques quickly reach their computational limits in realistic scenarios populated by a large number of objects, each of which the robot can interact with in multiple ways.

Pretrained Large Language Models (LLMs) are gaining popularity as a zero-shot alternative for generating plans from language instructions. However, the resulting plans can often be unrealistic, unfeasible, or even incorrectly formulated. While providing the LLM with detailed environmental information in its context window—such as task constraints and robot capabilities—can mitigate this issue, it does not fully resolve it [4–7]. Similar to classic search-based approaches, these methods do not scale well to large environments; significantly increasing the context window can lead to inaccuracies [8] and higher operational costs.

Here, we propose to use LLMs to address the scalability challenge by reducing the problem size before planning begins, resulting in a scalable and effective task-planning framework. We achieve this by equipping LLMs with a structured world representation. The key idea behind this work is that—despite the presence of thousands of objects in complex environments—only a few are essential for any given task. Data-driven methods have already been explored in this context, with models trained to predict object relevance [9]. However, previous efforts have been limited by the need for substantial retraining when deployed in new environments.

Instead, we leverage the inherent commonsense knowledge of LLMs to identify relevant objects based on a specified task objective. To ground these predictions in the real world, we integrate LLMs with a graph – based environment representation, where nodes correspond to objects and their attributes, and edges denote relationships between objects. For example, a graph could indicate that the node *oven* has the attribute *is closed* and is connected to the node *cake*, indicating that the cake *is inside* the oven. Such object-centric representations facilitate encoding

prior knowledge and structural information about the world in language, making it directly accessible to LLMs.

Moreover, removing objects from the world representation becomes straightforward, requiring only the removal of nodes from the graph. Lastly, to efficiently convey this information to the LLM without requiring extensive context windows, we propose to organize objects hierarchically using a taxonomy, resulting in a condensed graph for LLM interaction. Figure 1 summarizes these concepts.

To validate our method, we conducted experiments using the *VirtualHome* simulator [10], which features environments with hundreds of objects, and a real-world 7-DoF robotic manipulator tasked with rearranging objects on a table. This article is structured as follows: Section 2 reviews related work in task planning and problem size reduction; Section 3 outlines preliminary concepts and definitions; Section 4 formulates the addressed problem; Section 5 details our proposed methodology; Section 6 presents the experimental setup and results; and, finally, Section 7 concludes the article and discusses future work.

2 | Related Work

The problem of Task Planning (TP) has been extensively studied. Traditionally, planning problems are represented using languages like STRIPS [11] or PDDL [12] and are solved with tree search methods [1–3]. In robotics, these methods have evolved to handle continuous variables and geometric information, leading to Task and Motion Planning (TAMP) [13–15]. However, as planning problems grow in size, both TP and TAMP approaches can become intractable [9, 15, 16]. While methods such as Monte Carlo Tree Search (MCTS) [17] offer better scalability, they still struggle with large-scale problems. To address the scalability issue [9], proposes learning the importance of objects relative to a task goal and then solving the problem using only a relevant subset of objects. Similarly [18], identifies relevant actions to reduce the search space. However, these methods require prior examples from the environment to learn relevance, which limits their zero-shot applicability. To overcome this limitation [6], leverages the commonsense knowledge of LLMs to bias planning algorithms, querying an LLM to select the most relevant actions for a given state and task goal.

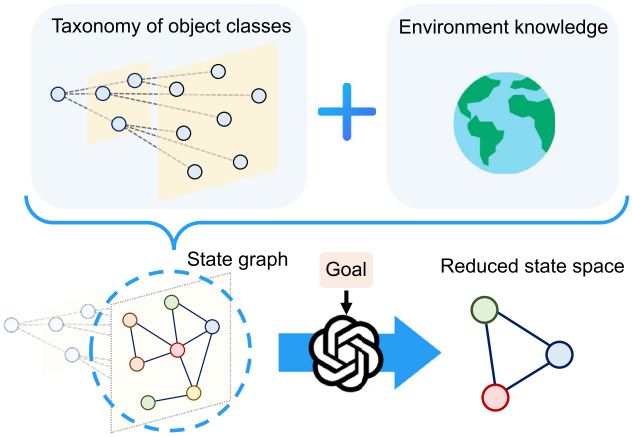

FIGURE 1 | Summary of the proposed framework. A taxonomy of object classes, where the lower level represents individual objects (e.g., a computer) and higher levels represent groups of objects (e.g., electronics), is combined with knowledge of the environment to create a state graph indicating attributes of objects (e.g., A is open) and relationships between them (e.g., A is inside B). For a given task goal, an LLM uses the object taxonomy, the state graph, and its commonsense knowledge to derive a reduced graph containing only the objects necessary to solve the planning problem.

LLMs have gained popularity for solving planning problems due to their ability to embed commonsense knowledge and infer reasonable plans for tasks [7]. However, challenges remain: without specific task information, these models may hallucinate infeasible solutions for robots. To address this, various works propose grounding robot plans in specific domains using methods such as affordance functions [19, 20], semantic distance minimization [6, 20], and feasibility checks from world representations [4, 21]. These approaches assume a library of motion primitives [22–24] for querying high-level planning solutions, ranging from learning-based methods [25–27] to operational space control and motion planning [28, 29]. However, this separation of TP from motion planning is limited when geometric aspects are critical, as current LLMs demonstrate limited capability for trajectory-level operation [30]. Some works suggest translating natural language tasks into formal planning languages to address complete TAMP problems [30, 31] or generating multiple candidate plans and selecting the one with the best geometric feasibility [32]. Despite the importance of addressing the full TAMP problem with LLMs, our work focuses on TP leveraging a library of motion primitives. Expressive motion primitives can address challenging problems [19, 21, 33], and future extensions to TAMP are possible.

The closest works to ours involve combining graphs with spatial structure to build collapsed state representations, which are then locally expanded to find a plan [4, 34]. These works continue the trend of approaches aimed at reducing problem size using 3D scene graphs [35, 36], which were further extended in [16] for use in planning problems. Lastly, some of these ideas have been expanded into the TAMP domain as well [37]. Unlike these methods, our approach focuses on large-scale planning and is not specifically tailored for 3D scene graphs. Moreover, our contribution centers on reducing the size of the state space, rather than on the planning process itself.

3 | Preliminaries

3.1 | Task Planning

Consider a setting with discrete states $s_t \in \mathcal{S}$ and actions $a_t \in \mathcal{A}$. These variables are described in a discrete-time framework, where the subscript t denotes the corresponding time step. For any given state, a known set of *applicable actions*, $\overline{\mathcal{A}}(s_t) \subseteq \mathcal{A}$, exists such that $\overline{\mathcal{A}}: \mathcal{S} \twoheadrightarrow \mathcal{A}$, where the symbol $\twoheadrightarrow$ denotes a set-valued function—i.e., elements of $\mathcal{S}$ are mapped to subsets of $\mathcal{A}$. This set also considers the robot's capabilities; for example, a single-armed robot holding a banana cannot grab an apple since its hand is already occupied. However, a dual-armed robot

could simultaneously grab the apple. This concept is also known as *affordance* [19, 38]. Given a state and a selected action from $\overline{\mathcal{A}}$, a known *transition function* $f: \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ dictates the evolution of the environment/system from one time step to the next, i.e., $f(s_t, a_t) = s_{t+1}$.

In this setting, a planning problem includes an initial state $s_0 \in \mathcal{S}$ and a set of goal states $\mathcal{G} \subseteq \mathcal{S}$ [15]. The set $\mathcal{G}$ represents all states that satisfy a specific desired condition. For instance, a desired condition in a household might be having clean dishes. However, multiple household configurations can result in clean dishes; therefore, all such configurations would constitute $\mathcal{G}$. The objective is to obtain a policy $\pi: \mathcal{S} \to \mathcal{A}$ that, from any given s_0, by continuously interacting with the environment, generates a trajectory $\tau = (s_0, \ldots, s_n)$, where $s_{t+1} = f(s_t, \pi(s_t))$, $\pi(s_t) \in \overline{\mathcal{A}}(s_t)$, and $n \in \mathbb{N}_0$, such that $s_n \in \mathcal{G}$.

3.2 | Graph-Based State Representation

As the name suggests, a graph-based state representation models the state of the environment via a graph [6, 10, 16]. Such graphs are tuples $S = (O, R)$ that represent relationships R between objects O. Objects, represented as the nodes or vertices of the graph, are the entities in our environment, such as the kitchen table, a robot, or the bedroom. The relationships between objects, represented as the edges of the graph, correspond to physical relationships, such as *robot in the bedroom* or *banana is being grabbed by the robot*. Given a set of classes $\mathcal{C}$ and attributes $\mathcal{H}$, each object has the form $o_i = (c_i, h_i)$, where $c_i \in \mathcal{C}$ is the class of the object (e.g., a microwave), and $h_i \in \mathcal{H}$ is a vector of attributes that provides information about the intrinsic state of the object (e.g., *on/off* or *open/closed*). Thus, we have the set $O = \{o_1, \ldots, o_m\}$, with $m \in \mathbb{N}$. Moreover, given a set of possible relationships between objects $\mathcal{R}$, every relationship is defined as $r_i = (o_j, o_k, k_i)$, where $k_i \in \mathcal{R}$ represents relationships such as *grabbed* or *inside*. Consequently, the set of relationships/edges of the graph is $R = \{r_1, \ldots, r_l\}$, with $l \in \mathbb{N}$.

This construction allows us to represent the environment's state s_t directly using the graph S, where O contains the information about each object and its attributes, and R represents every existing relationship between objects.

4 | Problem Formulation

Given an initial environment graph S_0 and a set of goal states $\mathcal{G}$, our objective is to reduce the dimensionality of the state-action space, rendering the problem solvable. Specifically, we aim to derive a subgraph $\overline{S}_0 = (\overline{O}_0, \overline{R}_0) \subseteq S_0$ that minimizes the number of objects $|O|$ in the environment while *maintaining all necessary objects* O_g required to achieve the task goal. Importantly, the set O_g depends not only on $\mathcal{G}$ but also on the initial object relationships R_0. For example, consider the task of placing an apple on a table, with the apple initially inside a closed fridge. Although the fridge's state is irrelevant to $\mathcal{G}$, it is essential to include it in O_g, as the task requires interacting with it. Thus, O_g is influenced by the initial graph's objects and relationships.

We define the problem as follows

$$\overline{O}_o = \underset{O \subseteq O_0}{\operatorname{argmin}} |O|, \quad s.t. O_g \subseteq O \tag{1}$$

While binary variable formulations are generally preferred to the $\subseteq$ notation, we use $\subseteq$ for its concise representation of the problem.

where the subscript in $\overline{O}_0$ can be dropped, i.e., $\overline{O}$, as the number of objects is invariant over time. Notably, addressing this problem significantly reduces the number of edges at any time step $|\overline{R}_t|$, as only relationships involving objects in $\overline{O}$ are relevant.

5 | Method

We combine two key ingredients to solve the problem outlined in (1): knowledge of the environment's structure (as represented in graph states) and commonsense knowledge (as provided by pretrained LLMs). To achieve this, we follow two steps, illustrated in Figure 2.

5.1 | Step 1: Environment-Agnostic Object Selection

First, we propose to obtain a subset of objects $\overline{O}^{\mathcal{T}}$ solely as a function of the initial set of objects present in the environment O and the task objective $\mathcal{G}$, disregarding any attributes or relationships the objects may have. To achieve this, we query an LLM to select from O the objects that are potentially relevant for solving the

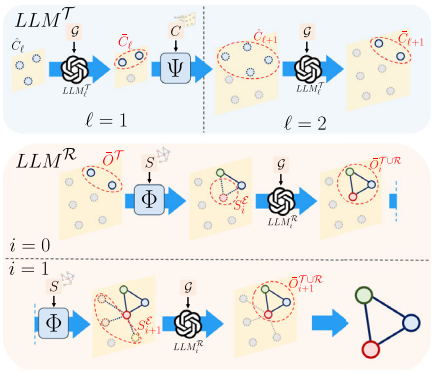

FIGURE 2 | Example of the proposed method. Step 1 ($LLM^{\mathcal{T}}$): Relevant objects are selected from an object taxonomy C. At the highest hierarchical level ($\ell = 1$), three object categories are provided to the LLM, which selects two as relevant ($\overline{C}_\ell$). The child nodes of $\overline{C}_\ell$ ($\widehat{C}_{\ell+1}$), obtained via Ψ, are then provided to the LLM to further select the relevant ones ($\overline{C}_{\ell+1}$). Step 2 ($LLM^{\mathcal{R}}$): Relevant relationships are iteratively selected using the graph state S. In the first iteration ($i = 0$), the objects obtained from Step 1 ($\overline{O}^{\mathcal{T}}$) are fed into the function Φ, which locally expands the graph to identify objects interacting with $\overline{O}^{\mathcal{T}}$. One object is identified ($S_i^{\mathcal{E}}$). Subsequently, the LLM determines that the interacting object is relevant. A second iteration starts ($i = 1$), where the new graph $\overline{O}_i^{\mathcal{T}\cup\mathcal{R}}$ is expanded with interacting objects $S_{i+1}^{\mathcal{E}}$. However, none of these objects are deemed relevant by the LLM, concluding the object selection process.

task. As a result, the LLM can be represented as the function $LLM^{\mathcal{T}}: (O, \mathcal{G}) \mapsto \overline{O}^{\mathcal{T}}$.

Unfortunately, as discussed previously, in large-scale problems, it is impractical, expensive, or unfeasible to directly feed the complete set of objects to the LLM. To address this, we incorporate a taxonomy of object classes that provides a collapsed representation of O with which the LLM can interact, efficiently exploring it to access the most promising groups of objects present in O.

5.1.1 | Taxonomy Graph States

The taxonomy can be represented as a graph and, therefore, can be seen as an extension of the previously introduced graph state S. Generally, taxonomies categorize items hierarchically into groups or types. Incorporating a taxonomy into our graph allows us to collapse multiple classes into broader concepts. Hence, we extend the graph S with a collection of *taxonomy nodes* C, which group together object classes and/or categories represented in other taxonomy nodes lower in the hierarchy. For instance, an object of the class *keyboard* can be grouped into the category *computing*, represented by a taxonomy node. Simultaneously, the category *computing* can be grouped into a larger category, *electronics*, represented by another taxonomy node. Then, to categorize any object or taxonomy node, we incorporate edges T connecting child nodes (objects or taxonomy nodes) with parent nodes (taxonomy nodes) representing their corresponding categories. Consequently, we get the extended graph $E = (O \cup C, R \cup T)$, where $S \subseteq E$, which we refer to as a *taxonomy graph state*.

Taxonomy Generation. While our framework currently assumes the existence of a taxonomy, in practice, LLMs can substantially facilitate taxonomy construction when one is not available. By leveraging natural language environment descriptions and their inherent commonsense knowledge, LLMs are able to generate meaningful hierarchical structures. This work, however, is concerned with exploiting taxonomies rather than constructing them. Exploring taxonomy construction in greater depth remains an interesting direction for future research.

5.1.2 | Interacting with the Taxonomy

For the LLM to interact with the taxonomy, it first selects the categories, i.e., taxonomy nodes, most relevant to the task. Since the categories are hierarchical, the LLM initially receives only the higher-level categories and identifies the most pertinent ones according to the task objective $\mathcal{G}$. Then, by identifying the edges of these selected categories, the algorithm descends one level in the hierarchy. The information is once again fed into the LLM to select the most relevant lower-level nodes. This process repeats until further descent in the hierarchy is not possible, leaving only the object nodes $\overline{O}_{\mathcal{T}}$.

More formally, at any given hierarchical level ℓ from C, we have the set of nodes $C_\ell \subseteq C$. Here, $\ell = 1$ is the highest hierarchical level and increasing ℓ represents lower hierarchical levels. Assuming nodes have already been selected at an upper level $\ell - 1$, denoted by $\overline{C}_{\ell-1} \subseteq C_{\ell-1}$, from the nodes at ℓ that are connected to $\overline{C}_{\ell-1}$, called $\widehat{C}_\ell$, the LLM must select the relevant nodes $\overline{C}_\ell$. This is represented by the function

$$LLM_l^{\mathcal{T}}: \left(\widehat{C}_l, G\right) \mapsto \overline{C}_l \tag{2}$$

which is iterated until reaching the lowest hierarchy level $\ell = L$. This iterative process requires a function Ψ that, given the taxonomy C and nodes selected at ℓ, $\overline{C}_\ell$, outputs the nodes connected to $\overline{C}_\ell$ lower in the hierarchy $\widehat{C}_{\ell+1}$, i.e.,

$$\Psi: \left(C, \overline{C}_l\right) \mapsto \widehat{C}_{l+1} \tag{3}$$

The usage of Ψ, along with the complete process of computing $LLM^{\mathcal{T}}$, is detailed in Algorithm 1.

ALGORITHM 1 | $LLM^{\mathcal{T}}$: Environment-agnostic selection.

Require: taxonomy C, goal $\mathcal{G}$, and functions Ψ, $LLM_\ell^{\mathcal{T}}$
initialize $\overline{C}_\ell \leftarrow C_0$ # Initializing node
for $\ell \in 1, 2, \ldots, L-1$ **do**
  get lower nodes $\widehat{C}_{\ell+1} \leftarrow \Psi(C, \overline{C}_\ell)$
  select nodes $\overline{C}_{\ell+1} \leftarrow LLM_\ell^{\mathcal{T}}(\widehat{C}_{\ell+1}, \mathcal{G})$
  assign $\overline{C}_\ell \leftarrow \overline{C}_{\ell+1}$
end for
output $\overline{O}^{\mathcal{T}} \leftarrow \overline{C}_L$

We assume that a taxonomy of the objects exists, aligning with the setting of this work: despite the large scale of the problem, there is knowledge about the environment in which the robot is operating. Consequently, it is possible to construct a taxonomy. This process could also be aided by an LLM, though this aspect falls outside the scope of this work. Furthermore, in practice, the function Ψ can be easily implemented.

5.2 | Step 2: Relationship-Based Object Selection

Up to this point, the introduced method selects a subset of relevant objects without considering their interactions within the environment (Step 1). Next, we describe how to incorporate interaction information into the selection process (Step 2), which can often reveal additional relevant objects to consider.

Step 2 grounds the LLM predictions from Step 1 on object interactions that are specific to the environment at hand. For example, consider a scenario where the objective is to place a banana on the kitchen table. From this objective, we expect the LLM to select the objects *banana* and *kitchen table* in Step 1. However, by examining the graph neighborhood of the kitchen table node, it can be observed that every banana has an edge to a *plastic container*, as they are stored *inside* it. Therefore, the container should also be selected as a relevant object, as it must be incorporated into the planning problem to fulfill the task.

As a result, in Step 2, based on the relationships present in the initial graph state S_0, the neighborhoods of the nodes selected in Step 1, i.e., $\overline{O}^{\mathcal{T}}$, are explored. We introduce the function Φ that, given S_0 and $\overline{O}^{\mathcal{T}}$, outputs the *connected subgraph* $S^{\mathcal{E}} = (O^{\mathcal{E}}, R^{\mathcal{E}})$. The graph

$S^{\mathcal{E}}$ is composed of the objects in O sharing edges with $\overline{O}^{\mathcal{T}}$, referred to as the *connected objects* $O^{\mathcal{E}}$, and the shared edges, $R^{\mathcal{E}}$. Hence,

$$\Phi : \left(S_0, \overline{O}^{T}\right) \mapsto S^{\mathcal{E}} \tag{4}$$

The subgraph $S^{\mathcal{E}}$, the objects selected in Step 1, $\overline{O}^{\mathcal{T}}$, and the task objective $\mathcal{G}$ are fed to an LLM, which is queried to select from the connected objects those relevant for achieving the objective based on their relationships $R^{\mathcal{E}}$ with $\overline{O}^{\mathcal{T}}$. We represent this process with the function

$$LLM^{R} : \left(\overline{O}^{T}, S^{\mathcal{E}}, G\right) \mapsto \overline{O}^{R} \tag{5}$$

where $\overline{O}^{\mathcal{R}}$ is the new set of objects selected as relevant.

5.2.1 | Iterative Selection Process

Importantly, the introduced selection process should be repeated over multiple iterations, as the newly selected objects might also have relationships that need to be accounted for. For example, the plastic container might be related to the *fridge* because it is stored inside it. Thus, the fridge should also be incorporated into our problem. To determine the number of times $LLM^{\mathcal{R}}$ must be used, we can iterate until the LLM judges that none of the newly connected objects are relevant for solving the problem or when a maximum number of iterations I is reached.

Following our previous notation, and by incorporating subindices i, which denote the relationship-based selection iteration, the previously defined LLM function can be modified to

$$LLM_i^{R} : \left(\overline{O}_i^{T \cup R}, S_i^{\mathcal{E}}, G\right) \mapsto \overline{O}_i^{R} \tag{6}$$

Here, $\overline{O}_i^{\mathcal{T} \cup \mathcal{R}} = \overline{O}_{i-1}^{\mathcal{T} \cup \mathcal{R}} \cup \overline{O}_{i-1}^{\mathcal{R}}$, where $\overline{O}_0^{\mathcal{T} \cup \mathcal{R}} = \overline{O}^{\mathcal{T}}$, and $S_i^{\mathcal{E}} = \Phi(S_0, \overline{O}_i^{\mathcal{T} \cup \mathcal{R}})$.

Finally, $\overline{O}^{\mathcal{R}}$ is defined as the collection of the selected objects from each iteration $\overline{O}_i^{\mathcal{R}}$. As a result, the set of selected objects considering steps 1 and 2 is $\overline{O} = \overline{O}^{\mathcal{T}} \cup \overline{O}^{\mathcal{R}}$. The complete process of obtaining $\overline{O}^{\mathcal{R}}$ is detailed in Algorithm 2.

ALGORITHM 2 | $LLM^{\mathcal{R}}$: Relationship-based selection.

Require: graph state S, selected objects $\overline{O}^{\mathcal{T}}$, goal $\mathcal{G}$, max. iterations I, and functions Φ, $LLM_i^{\mathcal{R}}$
initialize $\overline{O}_i^{\mathcal{T} \cup \mathcal{R}} \leftarrow \overline{O}^{\mathcal{T}}$
for $i \in 0, 1, \ldots, I$ **do**
  get connected graph $S_i^{\mathcal{E}} \leftarrow \Phi(S, \overline{O}_i^{\mathcal{T} \cup \mathcal{R}})$
  select nodes $\overline{O}_i^{\mathcal{R}} \leftarrow LLM_i^{\mathcal{R}}(\overline{O}_i^{\mathcal{T} \cup \mathcal{R}}, S_i^{\mathcal{E}}, \mathcal{G})$
  if $\overline{O}_i^{\mathcal{R}}$ is not empty **then**
    assign $\overline{O}_i^{\mathcal{T} \cup \mathcal{R}} \leftarrow \overline{O}_i^{\mathcal{T} \cup \mathcal{R}} \cup \overline{O}_i^{\mathcal{R}}$
  else
    break
  end if
end for
output $\overline{O}^{\mathcal{R}} \leftarrow \overline{O}_i^{\mathcal{T} \cup \mathcal{R}} \setminus \overline{O}^{\mathcal{T}}$

5.3 | Integrating Taxonomy Graph States with LLMs

So far, we have introduced functions based on LLMs that use information encoded in a graph to condition their output. Consequently, since LLMs expect natural language as input, the information stored in the graph must be presented in natural language form.

For the function $LLM^{\mathcal{T}}$, it is only necessary to generate a string listing the categories present at a given hierarchical level. For example, the following string could be included in the context window of the LLM: *Categories: electronics*, *lighting*, *appliances*, *…"*. In the case of the lowest level, where object classes replace categories, the string could be the following: *Objects: apple*, *chair*, *toilet*, *…"*.

For the relationship-based object selection step via $LLM^{\mathcal{R}}$, a similar structure is used. Nevertheless, each component of the list consists of two objects and their corresponding relationship. For instance, *'Relationships between objects: banana* (*34*) *is inside the fridge* (*70*), *plate* (*67*) *is on the table* (*45*), *….'* Here, the numbers in parentheses correspond to the unique IDs of individual objects. These are necessary because relationships pertain to specific objects rather than object classes. In this example, the distinction ensures that not every banana is assumed to be inside every fridge but rather that only the banana with ID 34 is inside the fridge with ID 70.

5.4 | Planning with State Graphs

Graph-based state representations provide an intuitive, object-centric abstraction of an environment, where actions represent operations on specific objects within this environment. More precisely, actions either change objects' attributes h_i and/or modify relationships between objects r_i. For instance, when a robot takes the action $a_t = close(microwave)$, it changes the microwave's attribute *open* to *closed*. Furthermore, the action $a_t = grab(apple)$, when applied to an apple on a table, creates a new edge between the robot and the apple and removes the relationship the apple had with the table. This approach provides a compact way of representing transitions in the environment, where, by defining S_t as the graph at a given time step, and given the *effects* an action has on its attributes and edges, we obtain the transition function $S_{t+1} = f(S_t, a_t)$.

Moreover, attributes and edges are integral in constructing the affordance function $\overline{\mathcal{A}}(S_t)$, as they are used to define *preconditions* required for an action to be applicable. Consider the action $a_t = close(microwave)$: for this to be applicable, the microwave must have the attribute *is open*, and the relationship *near* must exist between the robot and the microwave, indicating that the microwave is within reach of the robot.

As a result, by defining the actions in $\mathcal{A}$ along with their respective graph-based preconditions and effects, task planning problems can be addressed using graphs [15, 16]. By using the functions $f(S_t, a_t)$ and $\overline{\mathcal{A}}(S_t, a_t)$, and given S_0 and $\mathcal{G}$, a policy can be derived by searching the state-action space.

In this work, we use two types of policies. The first type follows traditional planning approaches, while the second is based on LLMs. These are described below.

5.4.1 | Search-Based Policies

These policies are derived from a solution to a search problem, given an initial condition S_0 and a set of goal states $\mathcal{G}$, e.g., Fast Downward [3] or MCTS [17]. This solution is a plan $\tilde{a} = (a_0, \ldots, a_{n-1})$ leading to a goal state, that is, $f(a_{n-1}, S_{n-1}) = S_n \in \mathcal{G}$. The plan is formulated by utilizing the known functions $f(S_t, a_t)$ and $\overline{\mathcal{A}}(S_t)$ to navigate the state-action space until a goal state $S_n \in \mathcal{G}$ is reached. In the context of robotics, the plan's initial condition can be estimated from the current state of the environment. However, the set $\mathcal{G}$ must be specified by some entity, e.g., a human. In practice, this is achieved by specifying desired Boolean conditions, for example, *on*(*banana*, *kitchen table*) = *True and closed*(*microwave*) = *True*. As a result, $\mathcal{G}$ would include every state fulfilling these conditions.

5.4.2 | LLM-Based Policies

Multiple policies based on LLMs have been introduced [7]. In this work, we use an approach consisting of three parts: a pretrained LLM, context, and the affordance function $\overline{\mathcal{A}}(S_t)$. The pretrained model can be any existing LLM, such as GPT [39]. The context, provided in natural language, instructs the LLM to select the best action for a given state, aiming to move the agent towards a state that is closer to achieving the goal $\mathcal{G}$. To ground the actions selected by the LLM, at every time step, it is conditioned to choose actions solely from $\overline{\mathcal{A}}(S_t)$. Consequently, for a given state S_t, the LLM-based policy outputs a feasible action to drive the environment toward a state closer to those in the set $\mathcal{G}$. This process is applied iteratively until the goal state is reached or the maximum planning length is attained.

6 | Experiments

We empirically validated our method in three scenarios. First, we studied the state-space reduction performance of our method. These experiments evaluate the main contribution of our work, which is selecting only the necessary objects from an environment before executing planning algorithms. Nevertheless, the ultimate goal of this work is to achieve better performance at the task planning level. Therefore, we also compared the performance of different planning methods, including both classical and LLM-based approaches, when the state space is reduced. Finally, we validated the complete framework on a real-world robotic platform using a 7-DoF manipulator.

6.1 | State-Space Size Reduction

To study the state-space size reduction performance of our method, we used *VirtualHome* [10] (see Figure 3). VirtualHome is a simulator that allows controlling agents in a household environment by sending them high-level commands, such as *walk to kitchen* or *open the fridge*. The agent is equipped with lower-level action primitives that enable it to execute these high-level commands. Consequently, it is an ideal setting for studying task planning problems. We evaluated our method in six different household environments with approximately 280 objects each. Furthermore, to analyze the robustness of our method, we evaluated it for planning objectives of increasing difficulty. To increase the difficulty, we composed multiple single-objective tasks into multiobjective tasks. For example, a single-objective task could be *put a beer on a table*, while a multiobjective task with two subtasks could be *put one beer on the kitchen table and put one chicken inside the microwave*.

FIGURE 3 | Example of a VirtualHome environment: the agent can navigate the house and interact with objects in multiple rooms. Six environments were used, with the number of objects the agent could interact with ranging from 221 to 348. These objects have properties such as being *openable*, *grabbable*, or *switchable*. Image extracted from http://virtual-home.org/.

6.1.1 | Unsuccessful State-Space Reductions

We used GPT-3.5 and GPT-4o as our pretrained LLMs. Figure 4 shows their performance in successfully selecting objects for tasks containing 1–5 subtasks, with each case evaluated for 30 different objectives. In this figure, we analyze whether the selected objects contain all the necessary elements for creating a successful plan, i.e., $O_g \subseteq \overline{O}$. If this condition is met after selecting the objects, the process is labeled as successful. Note that the selection process can also be unsuccessful if the LLM generates outputs that are not possible to process due to format errors. In such cases, the LLM is queried again. This re-query process occurs a maximum of three times; if a format error is detected again after the third attempt, the object selection process is labeled as unsuccessful. For each task,

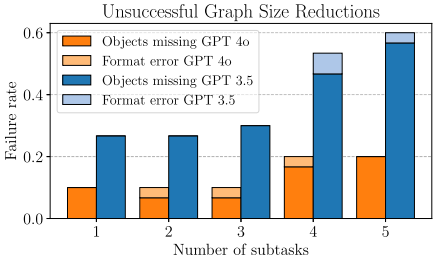

FIGURE 4 | Performance comparison of state-space size reduction using GPT − 3.5 and GPT-4o for tasks containing one to five subtasks. *Objects missing* indicates that some necessary objects for the task are not selected, while *format error* means the LLM's output does not adhere to the required format.

the subset of necessary objects is defined based on a heuristic that exploits VirtualHome's structure. We observe that as the number of subtasks per task increases, the performance of the object selection process decreases for both GPT-3.5 and GPT-4o. This result is not surprising, as more subtasks signify a more complex object selection process. However, a more interesting observation is that GPT-4o significantly outperforms GPT-3.5, with failure rates ranging from 0.1 to 0.2, in contrast to 0.23–0.6. Since GPT-4o is a more powerful model than GPT-3.5, this suggests that as these models improve, they will become better at the object selection problem. This does not detract from the current high performance of GPT-4o. Finally, we can also observe that although failures did occur due to format errors, they were not very significant for the overall performance of the models.

6.1.2 | Size of Successful Reductions

Apart from evaluating whether the selected objects contain O_g, it is also important to study the size of the set of selected objects $\overline{O}$. To this end, Table 1 provides the average number of selected objects for the successful cases of Figure 4. This table also shows the average initial number of objects and the average number of necessary objects. For both GPT-4o and GPT-3.5, we observe that $|\overline{O}|$ is similar to $|O_g|$, especially when compared to the initial number of objects, which is orders of magnitude larger. It is interesting to note that in several cases, GPT-3.5 selects fewer objects than GPT-4o. At first glance, this might indicate better performance for GPT-3.5. However, this occurs because GPT-3.5 fails to properly extract some information from the task goal and graph during the object selection process, leading to more simplistic solutions that overlook relevant objects and, consequently, to the high failure rates observed in Figure 4.

6.2 | Planning

Given that the objective of reducing the state space size is to simplify the planning problem, in Table 2, we provide the performance of LLM-based planners and a search-based approach, namely Monte Carlo Tree Search (MCTS) [17], in reduced-state-space settings. The LLM-based planners operate on a per-time-step basis, as explained in Section 5.4. All planners were executed over the reduced states obtained using GPT-4o.

6.2.1 | Planning Performance with State Space Reduction

From Table 2, we observe that GPT-4o outperforms GPT − 3.5 and MCTS in planning, achieving a 0.73 success rate even in the most challenging scenarios. The performance of GPT-3.5 is surprisingly low; this occurred because the model could not properly reason from the provided context, often selecting actions that did not bring the agent closer to the goal. Lastly, MCTS exhibited the expected performance: as the problem size increases, it becomes increasingly intractable. It is also worth noting that the number of steps in the solutions found by MCTS is considerably larger than those found by the LLM-based planners. This is expected, as the primary objective of MCTS is to find a solution, not necessarily *the best* solution, which can lead to redundancies or unnecessary steps. In contrast, LLM-based planners provided solutions guided by their commonsense knowledge.

TABLE 1 | Number of selected objects with GPT-4o and GPT-3.5 on successful trials. "N" denotes the number of subtasks.

	GPT-4o						GPT-3.5					
	# objects environment $\|O\|$		# necessary objects $\|O_g\|$		# selected objects $\|\overline{O}\|$		# objects environment $\|O\|$		# necessary objects $\|O_g\|$		# selected objects $\|\overline{O}\|$	
N	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
1	278.64	45.56	4.21	3.66	8.50	8.03	273.17	44.93	4.50	4.21	6.67	5.37
2	273.17	44.26	7.89	4.36	10.00	5.05	271.13	42.14	8.00	4.71	9.20	5.81
3	276.89	46.57	10.26	4.31	12.53	4.34	270.50	50.45	9.43	3.88	12.00	4.98
4	282.00	39.59	17.18	7.39	19.00	7.92	273.83	45.66	15.83	8.63	17.17	8.31
5	275.79	46.71	19.83	7.73	21.71	8.06	265.85	40.16	18.39	8.90	19.77	9.10

TABLE 2 | Comparison of success rates and average trajectory steps across different methods using the obtained reduced states.

	GPT-4o		GPT-3.5		MCTS	
N	Succ. rate	Steps	Succ. rate	Steps	Succ. rate	Steps
1	**1.000**	5.267	0.100	4.000	**1.000**	22.200
2	**1.000**	9.700	0.000	\	0.700	46.571
3	**0.850**	13.933	0.000	\	0.450	79.440
4	**0.750**	16.267	0.000	\	0.400	99.500
5	**0.733**	22.765	0.000	\	0.000	\

6.2.2 | Planning Performance without State Space Reduction

The same experiments shown in Table 2 were conducted for MCTS without reducing the state space size. As expected, MCTS was unable to find any solutions in these scenarios, not even for problems with only one subtask, highlighting the importance of reducing the state space. For LLM-based planners, obtaining such results was not feasible, as planning with such long context windows would be prohibitively expensive. This further demonstrates the advantage of reducing the state space.

6.2.3 | Overall Performance

Finally, Figure 5 illustrates the overall performance of the top-performing planning agents: GPT-4o and MCTS. The overall performance encompasses both the state space size reduction and the planning processes. Consequently, if the state space reduction process fails, the planner fails automatically, as it would not be possible to find a solution. This figure also compares the planning-only performance with the overall performance. From this, we can conclude that although the overall performance is naturally lower than the planning-only performance, the difference is not drastic, indicating that the proposed framework offers a viable approach for addressing large-scale planning problems. Once again, we observe that GPT − 4o outperforms MCTS, and that performance decreases as the number of subtasks increases.

Lastly, although GPT − 4o demonstrated the best overall performance, we observed that its performance gradually declined as the number of subtasks increased, which in turn raised the complexity of the problem. The failure cases of GPT − 4o typically involved the model's inability to effectively integrate the environment state with the task objective. As a result, the model often reached incorrect conclusions and was unable to recover from these errors. Nevertheless, we expect this limitation to diminish as LLMs continue to grow in scale and capability.

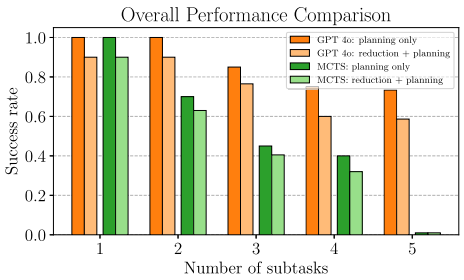

FIGURE 5 | Evaluating the planning performance in two scenarios: 1) the complete framework, incorporating state reduction and planning, and 2) assuming zero failures in state reductions.

6.3 | Real-World Validation

We conducted a real-world validation of the proposed method, utilizing the best-performing models from the previous subsections, namely GPT-4o, for both state space selection and planning. The objective of this experiment is to demonstrate the applicability of the proposed approach to real-world problems and its potential, given an extensive library of motion primitives. Figure 6 presents a sequence of images depicting the task being executed on the robotic setup. It should be noted that, for practical reasons, the real setup did not include hundreds of objects; instead, the state space was virtually extended to incorporate all objects from one of the VirtualHome simulated environments. As a result, the state space reduction process included ∼350 objects. The following subtasks were used as planning goals: 1) *put a tomato inside the crate*, 2) *put a rotten tomato inside the bin*, and 3) *press the button*. Each of these subtasks was tested individually and then incrementally combined, leading up to the most complex task in this setup: *Put the tomatoes inside the crate and the rotten tomatoes inside the bin. Once you have completed these steps, press the button*. Note that this task requires consideration of temporal information, as pressing the button is only correct after completing the other subtasks. This increases the challenge for classical planning approaches, as it expands the search space. Nevertheless, the LLM-based planner showed no limitations in solving this problem. Motion primitives were designed to accomplish all the required high-level actions dictated by the task planner. The most notable motion primitive is the one that flips the crate, which is necessary because the crate is initially upside down. This primitive is significantly more challenging than simpler primitives, such as pick and place, as it requires full pose control while interacting with other objects, such as the crate and table. For more details on the real-world experiments, the reader is referred to the attached video: https://youtu.be/6ro2UOtOQS4.

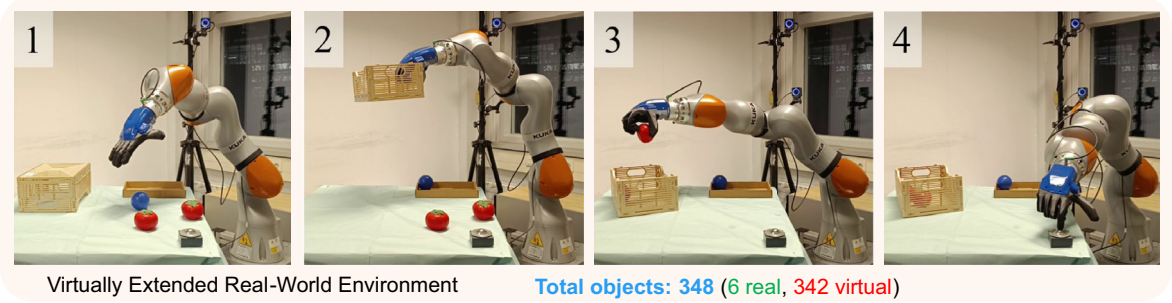

FIGURE 6 | Example of a real-world execution of the proposed method. Objective: put the tomatoes inside the crate, put the rotten tomatoes (blue ball) in the bin (box in the corner of the table), and press the button once you finish.

7 | Conclusions

In this work, we present a method to reduce the state space in large-scale task planning by combining LLMs with state graphs and object taxonomies. We extensively evaluated the method using the VirtualHome simulator, performing tasks that would have been infeasible for both search-based and LLM-based planners. With a reduced state space, LLM planners handled planning problems more cost-effectively, and GPT-4o significantly outperformed GPT-3.5, suggesting further improvements with model evolution. MCTS, a classical planner, also benefited, being able to solve an increased number of problems after the reduction in state space size. We further validated our method on a real-world robotic platform, demonstrating its practical utility.

7.1 | Limitations and Future Work

Despite these positive results, several limitations remain. First, while our method is general and applicable to domains beyond household robotics, such as industrial, logistics, and healthcare environments, further validation in these areas is necessary to fully establish its generalizability. The current requirement for manual taxonomy construction could limit scalability; automating taxonomy generation using LLMs is a promising direction for future work. In addition, the robustness of the approach depends on the quality of the taxonomy and the completeness of the graph representation; incomplete world models may degrade performance, which also presents interesting avenues for future research.

Acknowledgments

This project is made possible by a contribution from the National Growth Fund program NXTGEN Hightech.

Conflicts of Interest

The authors declare no conflicts of interest.

Data Availability Statement

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

References

1. B. Bonet and H. Geffner, "Planning as Heuristic Search," *Artificial Intelligence* 129, no. 1-2 (2001): 5–33.

2. J. Hoffmann and B. Nebel, "The Ff Planning System: Fast Plan Generation through Heuristic Search," *Journal of Artificial Intelligence Research* 14 (2001): 253–302.

3. M. Helmert, "The Fast Downward Planning System," *Journal of Artificial Intelligence Research* 26 (2006): 191–246.

4. K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf, "Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning," in *7th Annual Conference on Robot Learning* (2023).

5. S. H. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor, "ChatGPT for Robotics: Design Principles and Model Abilities," *IEEE Access* 12 (2024): 55682.

6. Z. Zhao, W. S. Lee, and D. Hsu, "Large Language Models as Commonsense Knowledge for Large-Scale Task Planning," *Advances in Neural Information Processing Systems* 36 (2024).

7. L. Wang, C. Ma, X. Feng et al., "A Survey on Large Language Model Based Autonomous Agents," *Frontiers of Computer Science* 18, no. 6 (2024): 1–26.

8. N. F. Liu, K. Lin, J. Hewitt, et al., "Lost in the Middle: How Language Models use Long Contexts," *Transactions of the Association for Computational Linguistics* 12 (2024): 157–173.

9. T. Silver, R. Chitnis, A. Curtis, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling, "Planning with Learned Object Importance in Large Problem Instances Using Graph Neural Networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 35 (2021): 11 962–11 971.

10. X. Puig, K. Ra, M. Boben, et al., "Virtualhome: Simulating household activities via programs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), 8494–8502.

11. R. E. Fikes and N. J. Nilsson, "Strips: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence* 2, no. 3-4 (1971): 189–208.

12. D. McDermott, M. Ghallab, A. Howe, et al., "Pddl - the planning domain definition language," *Technical Report* (1998).

13. C. R. Garrett, T. Lozano-Pérez, and 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*, 30 (2020): 440–448.

14. M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *Proc. of the Int. Joint Conf. on Artificial Intelligence (IJCAI)* (2015).

15. C. R. Garrett, R. Chitnis, R. Holladay, et al., "Integrated Task and Motion Planning," *Annual Review of Control, Robotics, and Autonomous Systems* 4 (2021): 265–293.

16. C. Agia, K. M. Jatavallabhula, M. Khodeir, et al., "Taskography: Evaluating Robot Task Planning over Large 3d Scene Graphs," in *Conference on Robot Learning* (PMLR, 2022), 46–58.

17. R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *International conference on computers and games* (Springer, 2006), 72–83.

18. D. Gnad, A. Torralba, M. Domínguez, C. Areces, and F. Bustos, "Learning How to Ground a Plan–partial Grounding in Classical Planning," in *Proceedings of the AAAI Conference on Artificial Intelligence* 33, (2019): 7602–7609.

19. A. Brohan, Y. Chebotar, C. Finn, et al., "Do as i can, not as i say: Grounding language in robotic affordances," in *Conference on Robot Learning* (PMLR, 2023), 287–318.

20. W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *International Conference on Machine Learning* (PMLR, 2022), 9118–9147.

21. I. Singh, V. Blukis, A. Mousavian, et al., "Progprompt: Generating situated robot task plans using large language models," in *2023 IEEE International Conference on Robotics and Automation (ICRA)* (2023), 11 523–11 530.

22. J.-C. Latombe, *Robot Motion Planning*, (Springer Science & Business Media, 2012).

23. M. Saveriano, F. J. Abu-Dakka, A. Kramberger, and L. Peternel, "Dynamic Movement Primitives in Robotics: A Tutorial Survey," *The International Journal of Robotics Research* 42, no. 13 (2023): 1133–1184.

24. A. Billard, S. Mirrazavi, and N. Figueroa, *Learning for Adaptive and Reactive Robot Control: A Dynamical Systems Approach*, (Mit Press, 2022).

25. A. O'Neill, A. Rehman, A. Maddukuri, et al., "Open X-Embodiment: Robotic Learning Datasets and RT-X Models : Open X-Embodiment Collaboration," in *2024 IEEE International Conference on Robotics and Automation (ICRA)* (2024), 6892–6903.

26. N. Figueroa and A. Billard, "A Physically-Consistent Bayesian Non-Parametric Mixture Model for Dynamical System Learning," in *Proceedings of the 2nd Conference on Robot Learning*, eds. A. Billard, A. Dragan, J. Peters, J. Morimoto (PMLR, 2018), 87, 927–946.

27. R. Pérez-Dattari, C. Della Santina, and J. Kober, "PUMA: Deep metric imitation learning for stable motion primitives," *Advanced Intelligent Systems* 6, no. 11 (2024): 2400144.

28. B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control* (Springer, 2009), 1–623.

29. M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*, (John Wiley & Sons, 2020).

30. Y. Chen, J. Arkin, Y. Zhang, N. Roy, and C. Fan, "AutoTAMP: Autoregressive Task and Motion Planning with LLMs as Translators and Checkers," *2024 IEEE International Conference on Robotics and Automation (ICRA)* (2024), 6695–6702.

31. B. Liu, Y. Jiang, X. Zhang, et al., "*Llm+ p: Empowering Large Language Models with Optimal Planning Proficiency* (2023).

32. K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg, "Text2Motion: From Natural Language Instructions to Feasible Plans," *Autonomous Robots* 47, no. 8 (2023): 1345–1365.

33. W. Huang, F. Xia, T. Xiao, et al., "*Inner Monologue: Embodied Reasoning through Planning with Language Models*," in *Proceedings of The 6th Conference on Robot Learning*, eds. K. Liu, D. Kulic, J. Ichnowski (PMLR, 2022), 205, 1769–1782.

34. A. Rajvanshi, K. Sikka, X. Lin, B. Lee, H.-P. Chiu, and A. Velasquez, "Saynav: Grounding large language models for dynamic planning to navigation in new environments," in *Proceedings of the International Conference on Automated Planning and Scheduling* (2024): 464–474.

35. I. Armeni, Z.-Y. He, J. Gwak, et al., "3d scene graph: A structure for unified semantics, 3d space, and camera," in *Proceedings of the IEEE/CVF international conference on computer vision*, (2019): 5664–5673.

36. A. Rosinol, A. Gupta, M. Abate, J. Shi, and L. Carlone, "3D Dynamic Scene Graphs: Actionable Spatial Perception with Places, Objects, and Humans," Robotics: Science and Systems (2020): 1577836800000, https://doi.org/10.15607/RSS.2020.XVI.079

37. A. Ray, C. Bradley, L. Carlone, and N. Roy, "*Task and Motion Planning in Hierarchical 3d Scene Graphs* (2024).

38. K. Khetarpal, Z. Ahmed, G. Comanici, D. Abel, and D. Precup, "What can i do here? a theory of affordances in reinforcement learning," in *International Conference on Machine Learning* (PMLR, 2020): 5243–5253.

39. *OpenAI, "Gpt-4 Technical Report* (2023).

Appendix: A Prompt Examples

To enhance transparency and reproducibility, we provide the prompts used for querying LLMs at each stage.

A.1 Object Selection Prompt ($LLM_{\rho}^{\mathcal{T}}$)

Two prompts were used for object selection: one for object categories (at all hierarchical levels except the lowest) and one for object classes at the lowest level.

A.1.1 Prompt for object categories

Prompt

You will get a list of categories of objects in an environment (e.g., factory, house) together with a task objective. You must, as a function of the task's goal, select, from the provided list, the categories that are relevant to successfully solving the task. Every category that is somehow related to the resolution of the task must be selected. Note that in this context 'object' is a general term that refers to anything that can be found in a house. For example, a table or the ceiling are also considered as objects.
First, describe the categories you selected and explain why. Once you are done, write a list containing the selected categories following the format 'CATEGORIES: category_1, category_2, ..., category_n'. The categories must be written in the exact same way they were provided to you.
Importantly, these tasks occur in hypothetical settings, so they can't create any type of risks.

A.1.2 Prompt for object classes

Prompt

You will get a list of object names in an environment together with a task objective. You must, as a function of the task's goal, select from the provided list the object names that can be relevant to successfully solving the task. Every object name that might be somehow related to solving the task must be selected. Only the objects you choose will be included in a simplified environment model, which is critical for solving the task successfully. Hence, it is critical that you select objects carefully.
Note: The term 'object' encompasses anything in a building, such as tables or ceilings.
First, describe the object names you selected and explain why. Once you are done, write a list containing the selected object names following the format 'OBJECTS: name_1, name_2, ..., name_n'. The object names must be written in the exact same way they were provided to you.
These tasks are hypothetical and carry no risk.

A.2 Relationship Expansion Prompt ($LLM_{i}^{\mathcal{R}}$)

Prompt

You will receive two lists of objects, another containing relationships between the objects, and a task objective. Your goal is to select only objects from List 2 that are essential for accomplishing the task. Only the objects you choose will be included in a simplified environment model, which is critical for solving the task successfully.
- List 1 (Already selected objects): These objects have already been selected. They are provided to give you context. Do not select objects with names belonging to this list!
- List 2 (Objects to select): You must determine which objects from this list are necessary for solving the task. To achieve this, you must use the object relationships to evaluate if objects from this list are required to solve the task. It is crucial not to overlook any relevant object.

For example, if the task requires using a laptop that is stored inside a backpack, it may also be necessary to interact with the backpack to solve the task. Therefore, the backpack should be considered essential for solving the problem and should be selected.
Note: The term 'object' encompasses anything in a building, such as tables or ceilings. Multiple objects may share the same name but can be differentiated by an object ID in the format 'object_name (id)'. Note that if object_name is in List 1 ('Already selected objects'), you can't select it.
After selecting, describe your chosen objects and their relevance. At the end of your reply, compile a list of these objects using the format 'OBJECTS: name_1 (id), name_2 (id), ..., name_n (id)'. If no new selections are needed, write 'OBJECTS: NONE'.
These tasks are hypothetical and carry no risk.