L3M+P: Lifelong Planning with Large Language Models

Krish Agarwal*†, Yuqian Jiang*†, Jiaheng Hu†, Bo Liu†, Peter Stone†§

***Abstract*— By combining classical planning methods with large language models (LLMs), recent research such as LLM+P has enabled agents to plan for general tasks given in natural language. However, scaling these methods to general-purpose service robots remains challenging: (1) classical planning algorithms generally require a detailed and consistent specification of the environment, which is not always readily available; and (2) existing frameworks mainly focus on isolated planning tasks, whereas robots are often meant to serve in long-term continuous deployments, and therefore must maintain a dynamic memory of the environment which can be updated with multi-modal inputs and extracted as planning knowledge for future tasks. To address these two issues, this paper introduces L3M+P (Lifelong LLM+P), a framework that uses an external knowledge graph as a representation of the world state. The graph can be updated from multiple sources of information, including sensory input and natural language interactions with humans. L3M+P enforces rules for the expected format of the absolute world state graph to maintain consistency between graph updates. At planning time, given a natural language description of a task, L3M+P retrieves context from the knowledge graph and generates a problem definition for classical planners. Evaluated on household robot simulators and on a real-world service robot, L3M+P achieves significant improvement over baseline methods both on accurately registering natural language state changes and on correctly generating plans, thanks to the knowledge graph retrieval and verification.**

I. INTRODUCTION

Large language models (LLMs) have proven to be very promising natural language (NL) interfaces in a variety of domains, including Robotics [1]. However, their lack of grounding in the physical world prevents them from being used effectively as direct planners in "agentic" systems [2]. A large body of work has thus been motivated toward grounding LLM-based agents to bridge the gap between human-friendly interfaces and consistent, accurate planning [3], [4], [5], [6], [7], [8].

However, applying these recent advancements to general-purpose service robots remains difficult. A primary challenge is that these robots are meant to serve their function over an extended period of time, and any actions they take must be grounded in a *dynamic*, real-world environment. We can consider the following two cases to see why an LLM alone is insufficient for interfacing with a service robot, and an external structured memory is required.

Suppose a human in the environment informs the service robot on events that have taken place in the environment. These events affect the state of the environment, in turn affecting future planning performed by the service robot. An LLM-only solution might keep track of these events by accumulating them into a report and feeding this report as context to the LLM whenever a planning query is processed. However, the report can grow arbitrarily long, so this approach is prone to model hallucination, which can significantly affect the accuracy of the agent [9]. As such, the LLM-based agent must have an external memory.

Of course, it is unreasonable to expect a human to report every event that takes place in an environment. Specifically, a service robot can be expected to consume not only human dialogue but also sensor input to gain knowledge about the environment. Solutions already exist for traditional knowledge representation in service robots that can be updated based on sensor input [10], [11], [12]. In order to use robot perception alongside human dialogue, an LLM must be able to interact with a unified memory that is compatible with a traditional knowledge representation system.

This motivation leads us to develop the L3M+P framework for augmenting existing research on grounded LLM-based agents with a dynamic, structured memory. L3M+P interfaces with this memory as follows.

1) It uses a LLM-based natural language interface for updating the memory given NL descriptions of environment updates.
2) It integrates with robot perception so the memory can be updated based on sensory input.
3) It retrieves relevant information from the memory that can be used as context within an existing LLM-based, grounded planner.

The rest of this paper is organized as follows: Section II provides preliminary information for the modules in L3M+P. Section III introduces the formalisms used in this paper. Section IV explains the implementation of each component of L3M+P. Section V discusses experiments and results. Finally, Section VI highlights recent work related to L3M+P.

II. BACKGROUND

A. Planning with Language Models

LLM+P [4] is a framework for combining LLMs with classical planning to bridge the gap between NL task descriptions and symbolic planners. These symbolic planners operate using the planning domain definition language (PDDL), a commonly used language to formally define environments and tasks [13], [14]. A domain PDDL file defines the state-action space through (a) a set of predicates that can fully represent a state and (b) a set of actions for manipulating

*Equal contribution.

†Department of Computer Science, The University of Texas at Austin {krishagarwal, jiangyuqian, jiahengh}@utexas,edu, {bliu, pstone}@cs.utexas.edu

§Sony AI

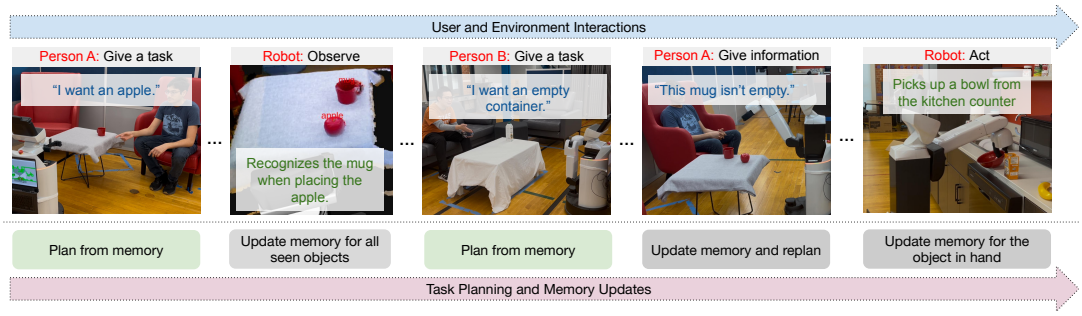


Fig. 1: L3M+P enables a household robot to keep a dynamic memory of verbal interactions and other sensory inputs and plan for long-horizon natural language tasks. Top: a sequence of robot interactions with the environment and users. Bottom: task planning and memory updates.

the current state. A problem PDDL file specifies a set of initial predicates (representing the initial state) and a set of goal conditions (identifying one or more goal states). Given a description of the initial state, a description of a task, and a domain file, LLM+P prompts an LLM to generate a problem PDDL file that can be supplied with the domain file to a symbolic planner to generate a plan.

L3M+P extends LLM+P by removing the requirement for a description of the initial state and instead maintaining the current state in a knowledge base that can be used to extract information relevant to a given planning task.

B. Knowledge Graphs

Knowledge graphs [15] are representations that organize information into graph structures, where nodes represent entities and edges represent relationships between entities. A knowledge graph can be constructed from various sources, including structured databases, unstructured text, or other forms of data. Similar to traditional databases, graph databases also generally support structured querying, such as through the Cypher query language [16].

The relationship between a pair of nodes in a knowledge graph is commonly referred to as a triplet of subject, predicate, and object. We work with the following triplet forms:

- `(subject, relationship, object)`
- `(subject, property, boolean)`

L3M+P uses a knowledge graph to represent the world state for a general-purpose robot.

C. Retrieval-Augmented Generation

Many applications of LLMs involve answering domain-specific queries. One approach for adapting a pretrained model to this purpose could be to fine-tune the model on domain-specific knowledge [17]. However, this method can be expensive and is not versatile to changes in knowledge. Retrieval-augmented generation (RAG) is a method for augmenting language models with external knowledge sources [18]. Context from an external knowledge source can be retrieved based on a user-prompt and fed alongside the original prompt to the language model in order to provide the language model with sufficient information to answer the prompt. Our framework has a RAG component which retrieves the relevant edges from a knowledge graph and feeds them as context for the LLM to handle state changes and plan queries.

III. PROBLEM STATEMENT

Our framework aims to solve two distinct but interdependent types of problems: keeping the robot's world state up-to-date, and generating plans based on the current world state.

A. World State Update

The first problem is to enable the robot to register updates that take place in the environment at any time during its operation. An update is provided either through natural language or through perception, and could represent a single event, multiple events, or even partial knowledge of events.

Formally, the input to an update problem U is the tuple $\langle \mathcal{S}, s_t, u_t \rangle$:

- $\mathcal{S}$ is a finite, discrete set of states that represent all possible world states.
- $s_t \in \mathcal{S}$ is the robot's knowledge at time t. s_t is itself the tuple $\langle V_t, E_t \rangle$, where V_t is a set of entities (vertices) in the environment and E_t is a set of relationships (edges) between the entities at time t.
- u_t is some verbal/sensory input describing an update taking place at time t.

Example World State Update Problem

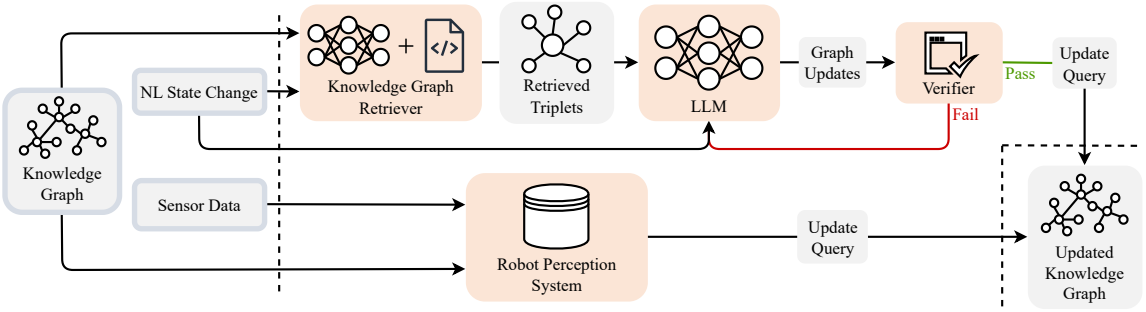
State Change: Gary went to Alexander's bedroom and placed the red pen on the table.

Correct Output:

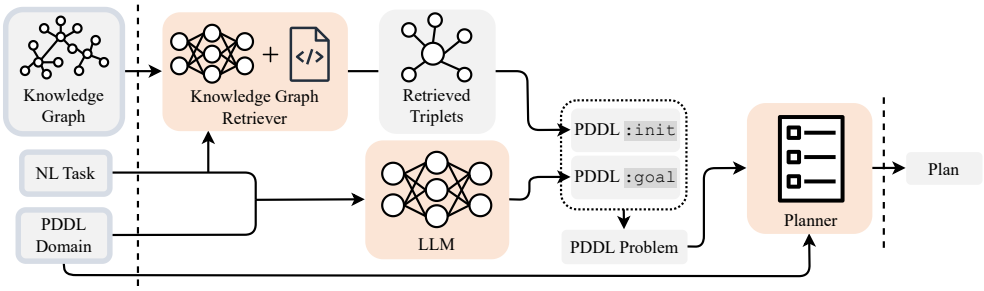
```
REMOVE: (red_pen, in_person_hand, gary),
(gary, person_in_room, jessica_bedroom)
ADD: (gary, person_in_room,
alexander_bedroom), (red_pen, placed_at_table,
alexander_bedroom_table)
```

Explanation: Gary moved from Jessica's bedroom to Alexander's bedroom. The red pen was placed on the table, leaving Gary's hand.

A correct solution to the problem U is the state $s_{t+1} \in \mathcal{S}$ that accurately represents the world state after registering the



(a) World State Updates



(b) Plan Generation

Fig. 2: (a) L3M+P keeps its knowledge graph consistent with the real world by receiving sensory input as well as descriptions of changes to the environment from humans. Both sources are used to update the knowledge graph to reflect an up-to-date representation of the environment. Verification is only necessary for Natural Language (NL) updates to guard against LLM hallucinations. (b) L3M+P uses the external knowledge graph to gather relevant context about the state of the environment to integrate with LLM+P for planning without requiring an explicit description of initial state for every planning query.

update u_t. If we assume that entities are not added/removed from the environment, the problem can be simplified to only output the change in the relationships $\langle E_t^-, E_t^+ \rangle$, where we interpret $E_{t+1} = E_t - E_t^- + E_t^+$, so $s_{t+1} = \langle V_t, E_{t+1} \rangle$.

B. Plan Generation

The second problem is to generate a plan to solve a given task. Formally, the input to the planning problem P is the tuple $\langle \mathcal{S}, s_t, g_t, \mathcal{A}, f \rangle$:

- $\mathcal{S}$ is again the state space.
- $s_t \in \mathcal{S}$ is again the robot's knowledge at time t.
- g_t is an NL description of a task to be completed by the robot at time t.
- $\mathcal{A}$ is a set of symbolic actions.
- $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the underlying state transition function. f takes the current state and an action and outputs the resulting state.

Example Plan Generation Problem

Task: Turn off the faucet in the bathroom.

Correct Output:

```
1) (move_to_room robot living_room bathroom)
2) (turn_off_faucet bathroom_sink bathroom
robot)
```

Explanation: The robot has to move from the living room to the bathroom. It can then turn off the faucet.

A solution to the planning problem P is a sequential plan $\pi = \langle a_1, a_2, \ldots, a_N \rangle$ with $a_i \in \mathcal{A}$.

IV. METHOD

We now describe how L3M+P solves the two problems defined above, as shown in Fig. 2.

A. Knowledge Graph Retrieval

To represent the state of the environment, L3M+P employs a knowledge graph (KG) that functions as a direct memory base for the agent. The nodes in the graph represent various entities in the environment and edges represent relationships between the entities.

The KG triplets have a one-to-one correspondence with PDDL predicates specified by the domain file. The KG is therefore a full representation of the current state $s \in \mathcal{S}$. It is important to note that we assume the system is provided a domain PDDL file that specifies predicates representing the environment state and actions the agent is allowed to take. This is a reasonable assumption to make since the process of creating this domain PDDL file is a one-time investment, after which the agent is able to act autonomously over the long-term.

To provide the RAG context for each update or plan query, the relevant nodes and edges are retrieved from the KG. One approach in current RAG pipelines is supplying a list of entities in the environment and prompting the LLM to select relevant entities based on a provided user prompt.

Algorithm 1 Search-Based Retrieval Algorithm

Input: V_t, E_t, entities and relationships in the knowledge graph at timestep t
Input: u_t or g_t, the verbal update or plan query
Output: V_r, relevant entities

1: Prompt the LLM to extract a query graph $< V_q, E_q >$, of ungrounded entities and relationships, from u_t or g_t
2: Compute the semantic similarity matrix S between V_q and V_t
3: Sort the entities in V_q by their highest similarity scores
4: $M_{optimal} \leftarrow$ **DFS**$(S, V_q, E_q, V_t, E_t, \{\}, \{\})$
5: $V_r \leftarrow$ entities mapped by $M_{optimal}$

Algorithm 2 Subgraph Matching Algorithm (DFS)

Input: S, node similarity matrix
Input: V_q, E_q, V_t, E_t
Input: M, M_+, current mapping and the best mapping found
Input: *cutoff*, a parameter for node similarity cutoffs
Output: $M_{optimal}$, mapping from V_q to V_t with the highest subgraph similarity score

1: **if** All nodes in V_q are mapped by M **then**
2: **if mapping_score**$(M) >$ **mapping_score**(M_+) **then**
3: **return** $M_{optimal} \leftarrow M$
4: **end if**
5: **return** $M_{optimal} \leftarrow M_+$
6: **end if**
7: $v_q \leftarrow$ pop front of V_q
8: Find all entity candidates $v_c \in V_t$ that $S[v_q, v_c] >$ *cutoff* $\times\, max_v(S[v_q, v])$
9: **for each** candidate v_c **do**
10: Add mapping $v_q \rightarrow v_c$ to M
11: $M_+ \leftarrow$ **DFS**$(S, V_q, E_q, V_t, E_t, M, M_+)$
12: Pop mapping $v_q \rightarrow v_c$ from M
13: **end for**
14: **return** $M_{optimal} \leftarrow M_+$

Then, given a list of relevant entities, all triplets outgoing and incoming to each relevant entity can be extracted at a certain maximum depth. These retrieved triplets function as the context for the downstream task.

When specific objects are described using attributes, the LLM may struggle to select the correct entities based on their names in the KG. We further propose a search-based retrieval method (Algorithm 1) to precisely locate entities that match the natural-language description. First, the algorithm prompts the LLM to generate a query graph with entities and their inferred relations from the natural-language query, and then searches for the most similar grounded sub-graph in the KG. The sub-graph matching algorithm (Algorithm 2) is inspired by recent work on scene graphs that supports querying a single central node of a sub-graph [19], [20]. Algorithm 2 uses depth-first-search (DFS) to find all entities that are relevant to a state update or task.

B. Updates to Knowledge Graph

L3M+P dynamically updates the knowledge graph when external changes in the environment are provided to the agent (shown in Fig. 2a). The updated knowledge graph can be used for re-planning a current task right away, or for solving other tasks including question answering. The agent can be alerted of these changes in two manners: (1) the robot receives sensory inputs, or (2) a human describes an event in the environment to the agent in natural language. We assume the robot has a perception system that converts observations into a representation (e.g. scene graphs) to update corresponding sub-graphs in the KG, and we show an example of such a system in the robot demo. As such, in this section we focus on NL descriptions of updates to the environment.

Any descriptions of environment changes need to be reflected in the KG to maintain a consistent memory base. Given such a description, the procedure defined by Algorithm 3 is followed. In summary, L3M+P

1) Retrieves a relevant subgraph from the KG (line 1)
2) Prompts the LLM to generate KG updates given the retrieved subgraph and the update description (line 5)
3) Verifies the generated graph updates against the domain PDDL (line 7)
4) Retries if the verifier fails (line 9)
5) Applies the LLM-generated graph updates when verification succeeds (lines 10 and 11)

L3M+P does not insert new entities into the knowledge graph from NL updates. The knowledge graph only tracks the instances of concrete objects seen by the robot.

Algorithm 3 NL Knowledge Graph Updates

Input: $\langle V_t, E_t \rangle$, the current knowledge graph at time t
Input: u_t, the NL update at time t
Output: E_{t+1}, the updated KG edges (the updated graph is $\langle V_t, E_{t+1} \rangle$)

1: $E_t^{rel} \leftarrow$ **retriever**(V_t, E_t, u_t) {Retrieve a relevant set of edges from the KG}
2: $E_t^{irrel} \leftarrow E_t - E_t^{rel}$
3: $prompt \leftarrow [V_t, E_t^{rel}, u_t]$
4: **repeat**
5: $output \leftarrow$ **LLM**$(prompt)$ {Prompt the LLM to generate a KG update}
6: $E_t^+, E_t^- \leftarrow$ **parse**$(output)$
7: $errors \leftarrow$ **verify**(E_t^+, E_t^-, E_t, V_t)
8: $prompt \leftarrow prompt + [errors]$
9: **until** $errors = \emptyset$ {Re-prompt LLM until generated update is valid}
10: $E_{t+1}^{rel} \leftarrow E_t^{rel} - E_t^- + E_t^+$
11: $E_{t+1} \leftarrow E_t^{irrel} + E_{t+1}^{rel}$
12: **return** E_{t+1}

C. Planning

L3M+P uses a modified version of the existing LLM+P framework to perform planning tasks (shown in Fig. 2b).

Algorithm 4 Planning

Input: $\langle V_t, E_t \rangle$, the current knowledge graph at time t
Input: g_t, the NL description for a task to complete at time t
Input: D, the PDDL domain
Output: $\pi = \langle a_1, a_2, \ldots, a_n \rangle$, a sequential plan to accomplish the given task

1: $E_t^{rel} \leftarrow$ **retriever**(V_t, E_t, g_t) {Retrieve a relevant set of edges from the KG}
2: $goal \leftarrow$ **LLM**(V_t, E_t^{rel}, D, g_t) {Prompt the LLM to generate a `:goal` block for the given task}
3: $P \leftarrow \{V_t, E_t^{rel}, goal\}$ {Generate a corresponding PDDL problem (E_t^{rel} populates the `:init` block)}
4: $\pi \leftarrow$ **planner**(D, P)
5: **return** π

In contrast to LLM+P, the user does not provide the full description of the environment. When presented with a task, L3M+P instead queries the KG to gain sufficient detail for solving the task (shown in Algorithm 4). In summary, L3M+P

1) Retrieves a relevant subgraph from the KG (line 1)
2) Prompts the LLM to generate the `:goal` block for the PDDL problem given the retrieved subgraph, the PDDL domain, and the task description (line 2)
3) Constructs a PDDL problem with the generated `:goal` block and using the retrieved subgraph as the `:init` block (line 3)
4) Passes the fully constructed PDDL problem to a symbolic planner alongside the provided PDDL domain to generate a plan (line 4)

V. EXPERIMENTS

Our experiments are designed to address the following questions:

1) Does RAG enable more accurate KG updates compared to just providing the full KG context to the LLM? In other words, does RAG allow for smaller prompts to prevent hallucinations, or does the lack of full context worsen performance? **(It provides a significant improvement)**
2) To what extent does the verification step improve (or possibly degrade) graph update accuracy? **(It provides a decent improvement)**
3) Does accurate KG state translate to correct plans, and conversely does incorrect KG state translate to failed plans? **(Yes)**
4) Does leveraging verbal updates in L3M+P improve the success rates of a service robot in solving tasks? **(Yes)**

To answer these questions, we present three types of experiments. Sec. V-A evaluates how L3M+P updates the KG based on NL state updates as well as generates plans based on a continuously updating KG. This text-based simulation assumes that updates occur at discrete time steps between tasks. Sec. V-B presents the results in an embodied simulator where the plan of the current task must be adapted to both verbal and sensory updates.[1] Sec. V-C demonstrates a robot successfully helping users in a home setting where correct KG updates are required to plan current and future tasks.

A. Text-Based Simulation

Since the goal is for the agent to work within a general-purpose service robot, we introduce an open-ended, text-based household simulator where items are randomly generated, humans randomly manipulate the environment, and random tasks are periodically presented to an agent. The household consists of a diverse set of items, such as tables, fridges, sinks, books, food items, dishes, lights, TVs, and phones, distributed across various rooms in the household such as a kitchen, living room, and bedrooms. Humans in the household may move objects, turn on/off any faucets/lights, answer phones, etc. The agent may be tasked with returning items to certain locations, washing dishes, providing items to humans (like food or drink), etc. The purpose of random tasks is to test the capabilities of the agent to function in a domain-agnostic environment.

The simulation provides the agent with an NL description of each environment update and an NL description of each planning task. The agent is also provided a domain PDDL file, which, as justified in Section IV, is reasonable for realistic use cases. To measure accuracy, the simulation maintains a ground-truth KG after each state update and plan execution to compare against L3M+P's proposed KG, and a ground-truth problem PDDL file is also created to generate a ground-truth plan that can be validated against any plans generated by the agent.

We test the following six variants of the agent against the same simulation:

1) R^-: The agent receives the full KG as context when performing state updates (no RAG).
2) R_V^-: The agent receives the full KG as state change context (no RAG). State updates are verified against the domain PDDL.
3) R^+: RAG is used to extract relevant context from the KG that fits in the LLM context window to perform state updates.
4) R_V^+: RAG is performed as above. State change verification is performed.
5) R^S: Search-based RAG is performed to extract KG context.
6) R_V^S (a.k.a L3M+P): Search-based RAG is performed as above. State change verification is performed.

To execute these experiments, we use OpenAI's GPT-4o language model. We also utilize the SIW-THEN-BFSF planner provided by LAPKT [21] as the planner in all experiments.

[1]The implementation of the simulation experiments is available at `https://github.com/krishagarwal/l3m-p.git`

Example of a Failed State Change and a Resulting Incorrect Plan Produced by R^-

State Change: Jessica turned off the overhead light in the laundry room.

Prompt: State Change + Full KG Context + Determine which of the relations should be removed and what new relations should be added...

GPT-4o (generated KG update):
```
REMOVE: (laundry_room_light, in_room,
laundry_room)
ADD: empty
```

Task: The water and electricity bills are high. Can you turn off all lights and faucets?

Retrieved KG context (`:init` block):
```
...(laundry_room_light, light_on, true)
```

Prompt: Task + Full KG Context + Provide the goal block for a problem PDDL file...

GPT-4o (generated `:goal` block):
```
(:goal (and (forall (?a - light) (not
(light_on ?a))) (forall (?b - sink) (not
(faucet_on ?b)))))
```

Plan:
```
...(turn_off_light laundry_room_light)
```

Example of a Successful State Change and Resulting Plan Produced by R_V^S

State Change: Kathleen placed the red pen on the 3rd level of the shelf in Jerry's bedroom.

Prompt: State Change + Retrieved KG Context + Determine which of the relations should be removed and what new relations should be added...

GPT-4o (generated KG update):
```
REMOVE: (red_pen, in_hand, kathleen)
ADD: (red_pen, placed_at_shelf,
jerry_bedroom_shelf), (red_pen,
on_shelf_level, shelf_level_3)
```

Task: Place the red pen on the 5th level of the shelf in Alexander's bedroom.

Retrieved KG context (`:init` block):
```
(red_pen, placed_at_shelf,
jerry_bedroom_shelf)
(red_pen, on_shelf_level,
shelf_level_3)...
```

Prompt: Task + Retrieved KG Context + Provide the goal block for a problem PDDL file...

GPT-4o (generated `:goal` block):
```
(:goal (and (on_shelf_level bisque_pen
shelf_level_5) (placed_at_shelf
bisque_pen alexander_bedroom_shelf)))
```

Plan:
```
...
(place_at_shelf red_pen
alexander_bedroom_shelf the_agent
alexander_bedroom shelf_level_5)
```

Here are our findings:

1) Based on Table I, RAG helps prevent model hallucinations to significantly increase update accuracy.
2) Syntactically verifying the state changes generated by the LLM against the PDDL domain also improves the accuracy of state changes. While the KG context generally informs the LLM on the proper syntax for KG triplets, verification serves as a stronger guarantee of correctness.
3) Knowledge graph accuracy translates to plan-generation accuracy. In most failure instances, the LLM-generated PPDL `:goal` block is accurate, but the planner produces an inaccurate plan or fails to even generate a plan due to an an inaccurate KG state (which forms the `:init` block of the PDDL problem). This leads to plans that (a) only partially solve the task or (b) cannot be executed due to their inconsistency with the true environment.
4) L3M+P is not perfect: the usual cause of failure is when KG retrieval does not include all relevant context for a given NL update or plan, so the LLM produces hallucinated output from insufficient information.
5) RAG enables a cost savings: As shown in Figure 3, RAG uses fewer tokens to process successful state changes. On average, R_V^S uses 67.6% fewer total input/output tokens compared to R^- to process NL state changes (including tokens used to perform RAG).
6) RAG also enables a time savings: As shown in Figure 4, R_V^S has a speedup when comparing (a) the combined RAG and planner time for R_V^S, against (b) only the planner time with the full KG context as the `:init` block (ground truth). For correct plans generated by R_V^S, this is an average speedup of 12.5x.

Success Rate % (↑)	Method					
	R^-	R_V^-	R^+	R_V^+	R^S	R_V^S (ours)
State Changes	71.5	71	77.5	92.5	96	**98**
Plans	72.5	72.5	80	85	87.5	**90**

TABLE I: Final state change and plan success rate % after running the six agent variants on the simulation.

B. Embodied Interactive Simulation

We compare task completion rates of L3M+P that incorporate both updates versus relying solely on robot perception in a variety of home tasks in the embodied AI simulator AI2-THOR [22]. The scenes are selected from the ProcTHOR-10k dataset [23]. The output of a robot perception system is simulated by providing the semantic properties of the

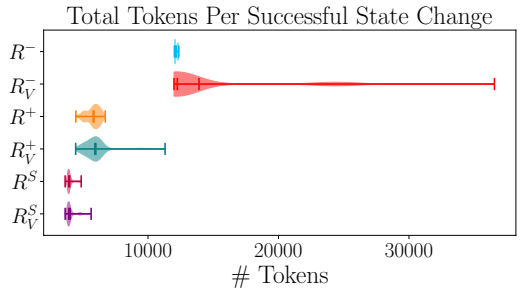


Fig. 3: Distributions of number of tokens used per successful state change in the text-based simulation. Variants that use RAG are the most token-efficient.

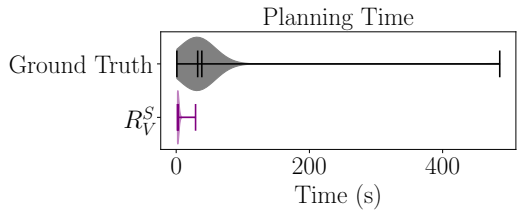


Fig. 4: Distributions of planner time for R^S_V (including time for RAG) and planner time against a ground truth PDDL with the full state provided, for tasks successfully solved by R^S_V. R^S_V is far more time efficient by only providing the planner with relevant context for each given task.

currently visible objects. The initial states and goal conditions are generated randomly for the following tasks: pick and place, wash dishes, clean bed, discard broken objects, and refrigerate food. State changes are scheduled randomly, and they can be categorized as either a reset (e.g. washed dishes getting dirty again), an addition (e.g. new dishes becoming dirty), or a relocation (e.g. dirty dishes are moved from their last known locations). For each condition, 20 task instances are generated. L3M+P re-plans when receiving verbal updates, and the visual-only variant re-plans when execution failures occur, indicating the environment is different from its knowledge at the last planning time. The results reported in Table II show much higher success rates when L3M+P leverages the verbal interactions. While visual-only performance could be improved by active perception behaviors to update knowledge, using verbal updates does not delay the task. Due to simulation uncertainties, some communicated updates do not align with the actual effects, leading to L3M+P failures.

Success Rate % (↑)	Task Type				
	Pick & Place	Wash Dishes	Clean Bed	Discard Broken	Keep Cold
L3M+P	65	85	90	70	65
Visual Only	15	0	5	10	55

TABLE II: Task completion rates on the 5 types of tasks by using L3M+P versus the variant with only visual updates.

C. Robot Demonstration

We present a robot demonstration in a scenario that builds upon the motivating example in Fig. 1. In the first task, Person A requests an apple. Initially, the robot plans to retrieve one from the fridge but instead picks up an apple spotted on the kitchen counter. Shortly after, Person B asks for a container to hold cereal. The robot has seen a mug on the table next to person A in the first task. However, before picking it up, Person A informs the robot that the mug is not empty. The robot then selects a bowl instead and retrieves the cereal from the previously scanned kitchen counter. Further details of the demonstration are available in the supplementary video.

VI. RELATED WORK

Several frameworks have already been proposed for LLM-based planning and reasoning [3], [24], [25], [7], [8]. While these frameworks demonstrate the potential of LLMs as plan generators, they may overestimate the extent to which language modeling capabilities directly translate to robust reasoning capabilities. Additionally, these works only focus on planning, and they do not attempt to design end-to-end solutions where the agent must both gather information about the environment and use that information to plan.

A few frameworks have been introduced to address planning within dynamic environments. LLM-DP [5] maintains a set of known information about the environment as well as a set of beliefs about the environment, which are gradually updated as the agent makes observations. The beliefs are used as context for an LLM to generate a PDDL world specification and goal for a classical planner. However, aside from the fact that this system was exclusively tested in Alfworld [26], a text-based environment that only hosts a very specific domain of problems, LLM-DP assumes that the only changes to the environment are caused by the agent and does not maintain knowledge of the environment across different planning tasks.

Statler [6] maintains and updates a world state to serve as a memory for an LLM-based agent across planning tasks. The current state, represented in a JSON-like format, is used as context for planning tasks and is updated based on performed actions. Similar to LLM-DP, this framework is unable to account for external changes to the environment. Additionally, a specific world-state format must be curated for different domains. Even if the world-state representation were made domain-agnostic, the system would break down in complex environments with many objects and relations, increasing the likelihood of LLM hallucination.

There has also been recent work on storing memory for general-purpose LLM agents. A-MEM [27] uses a flexible graph structure to organize information, populating nodes with atomic notes with assigned tags and dynamically inserting links between nodes based on similarity. A-MEM can be seen as an orthogonal approach to L3M+P for using a graph structure to maintain an agent memory–while A-MEM uses graph edges to loosely connect related information for more accurate retrieval, L3M+P uses graph edges to explicitly

represent relationships, and the benefits of accurate retrieval are a result of the graph structure.

Furthermore, Generative Agents [28] offers an approach to mimic human behavior with LLM agents in open-ended environments by incorporating dynamic memory, reflection, and planning capabilities. The agent records its experiences over extended periods in a natural language memory stream, from which information is retrieved to generate context-aware actions. This work addresses a separate problem from L3M+P, which is to allow agents to behave more creatively. As such, Generative Agents uses a loosely-defined memory, while L3M+P uses a structured and verified knowledge graph that is grounded in the real world to guarantee correctness.

VII. CONCLUSION

We propose L3M+P, a framework that extends LLM and planning systems to support lifelong deployments of general-purpose service robots. L3M+P maintains a dynamic memory base that can be continuously updated based on various, multi-modal sources of information about the environment, and remains in a consistent format that can be queried for writing planning specifications. At planning time, relevant context from this memory is extracted to generate accurate specifications about the environment and task, which can then be utilized by classical planners. The use of an external memory base enables representing the world state of fairly complex environments in a domain-agnostic manner, and the use of context retrieval prevents issues with limited context windows/hallucinations. In future work, the system could be enhanced to detect when re-planning is necessary in environments with frequent updates.

REFERENCES

[1] C. Zhang, J. Chen, J. Li, Y. Peng, and Z. Mao, "Large language models for human–robot interaction: A review," *Biomimetic Intelligence and Robotics*, vol. 3, no. 4, p. 100131, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667379723000451

[2] K. Mahowald, A. A. Ivanova, I. A. Blank, N. Kanwisher, J. B. Tenenbaum, and E. Fedorenko, "Dissociating language and thought in large language models," 2024.

[3] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng, "Do as i can, not as i say: Grounding language in robotic affordances," 2022.

[4] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone, "Llm+p: Empowering large language models with optimal planning proficiency," 2023.

[5] G. Dagan, F. Keller, and A. Lascarides, "Dynamic planning with a llm," 2023.

[6] T. Yoneda, J. Fang, P. Li, H. Zhang, T. Jiang, S. Lin, B. Picker, D. Yunis, H. Mei, and M. R. Walter, "Statler: State-maintaining language models for embodied reasoning," 2023.

[7] C. H. Song, J. Wu, C. Washington, B. M. Sadler, W.-L. Chao, and Y. Su, "Llm-planner: Few-shot grounded planning for embodied agents with large language models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2023, pp. 2998–3009.

[8] V. Bhat, A. U. Kaypak, P. Krishnamurthy, R. Karri, and F. Khorrami, "Grounding llms for robot task planning using closed-loop state feedback," 2024.

[9] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, and T. Liu, "A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions," 2023.

[10] Y. Breux and S. Druon, "Multimodal object-based environment representation for assistive robotics," *International Journal of Social Robotics*, vol. 12, no. 3, pp. 807–826, Jul 2020. [Online]. Available: https://doi.org/10.1007/s12369-019-00600-4

[11] S. R. Schmidt-Rohr, G. Dirschl, P. Meissner, and R. Dillmann, "A knowledge base for learning probabilistic decision making from human demonstrations by a multimodal service robot," in *2011 15th International Conference on Advanced Robotics (ICAR)*, 2011, pp. 235–240.

[12] Y. Jiang, N. Walker, J. Hart, and P. Stone, "Open-world reasoning for service robots," in *Proceedings of the 29th International Conference on Automated Planning and Scheduling (ICAPS 2019)*, July 2019.

[13] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl-the planning domain definition language," 1998.

[14] P. Haslum, N. Lipovetzky, D. Magazzeni, and C. Muise, "An introduction to the planning domain definition language," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, no. 2, pp. 1–187, 2019.

[15] L. Ehrlinger and W. Wöß, "Towards a definition of knowledge graphs," in *International Conference on Semantic Systems*, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:8536105

[16] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor, "Cypher: An Evolving Query Language for Property Graphs," pp. 1433–1445.

[17] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," 2022.

[18] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 9459–9474.

[19] H. Chang, K. Boyalakuntla, S. Lu, S. Cai, E. P. Jing, S. Keskar, S. Geng, A. Abbas, L. Zhou, K. Bekris, *et al.*, "Context-aware entity grounding with open-vocabulary 3d scene graphs," in *Conference on Robot Learning*. PMLR, 2023, pp. 1950–1974.

[20] S. Koch, N. Vaskevicius, M. Colosi, P. Hermosilla, and T. Ropinski, "Open3dsg: Open-vocabulary 3d scene graphs from point clouds with queryable objects and open-set relationships," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 14 183–14 193.

[21] M. Ramirez, N. Lipovetzky, and C. Muise, "Lightweight Automated Planning ToolKiT," http://lapkt.org/, 2015, accessed: 2020.

[22] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, M. Deitke, K. Ehsani, D. Gordon, Y. Zhu, A. Kembhavi, A. Gupta, and A. Farhadi, "AI2-THOR: An Interactive 3D Environment for Visual AI," *ArXiv*, vol. abs/1712.05474, 2017.

[23] M. Deitke, E. VanderBilt, A. Herrasti, L. Weihs, K. Ehsani, J. Salvador, W. Han, E. Kolve, A. Kembhavi, and R. Mottaghi, "Procthor: Large-scale embodied ai using procedural generation," *Advances in Neural Information Processing Systems*, vol. 35, pp. 5982–5994, 2022.

[24] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," 2022.

[25] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," 2023.

[26] M. Shridhar, X. Yuan, M.-A. Côté, Y. Bisk, A. Trischler, and M. Hausknecht, "Alfworld: Aligning text and embodied environments for interactive learning," 2021. [Online]. Available: https://arxiv.org/abs/2010.03768

[27] W. Xu, Z. Liang, K. Mei, H. Gao, J. Tan, and Y. Zhang, "A-mem: Agentic memory for llm agents," 2025. [Online]. Available: https://arxiv.org/abs/2502.12110

[28] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein, "Generative agents: Interactive simulacra of human behavior," 2023. [Online]. Available: https://arxiv.org/abs/2304.03442