L3M+P: Lifelong Optimal Planning with Large Language Models

Krish Agarwal[†], Yuqian Jiang[†], Jiaheng Hu[‡], Bo Liu[†], Peter Stone[†§]

***Abstract*— Large language models (LLMs) are demonstrated to excel in many generalization tasks but do not necessarily perform well as standalone reasoning agents in service robots. Recent research has been on integrating LLMs with classical planning methods so LLMs can serve as natural language interfaces to more robust underlying systems. The main limitation that classical planning brings is the requirement for detailed and consistent specifications of the environment. Additionally, service robots are meant to serve as long-term agents, so they must maintain a dynamic memory of the environment that they can (1) update based on multi-modal input and (2) consult with to perform planning tasks; current frameworks do not support this. To address these issues, this paper introduces L3M+P, a framework that uses an external knowledge graph as a memory base to represent an absolute world state. The graph can be updated from multiple sources of information, including natural language interactions with humans. L3M+P will enforce rules for the expected syntax of the absolute world state graph to maintain consistency between graph updates. At planning time, given a natural language description of a task, context will be retrieved from the knowledge graph and provided to the existing LLM+P framework, which will generate a problem specification in the form of a PDDL file that a classical planner can use to solve the task. Our experiments demonstrate that we can achieve 26.5% improvement on correctly registering natural language state changes and 17.5% improvement on generating correct plans by using both knowledge graph retrieval and rule verification on the knowledge graph.**

I. INTRODUCTION

Large language models (LLMs) have proven to be very promising natural language interfaces in a variety of domains, including in robot agents [1]. However, their lack of grounding in the physical world prevents them from being used effectively as direct planners in these robot systems [2]. This has motivated a large body of work for grounding LLM-based agents to bridge the gap between human-friendly interfaces and consistent, accurate planning.

With this area of research on frameworks that ground LLMs for planning comes the important question of how practical these systems really are. Service robots are the most direct application since they require exactly what these frameworks are solving: a human should be able to loosely state a task in natural language and the service robot should be able to generate and execute an accurate, grounded plan that accomplishes the task.

However, that is a very isolated example. While these frameworks address the fundamental goal of converting a natural language task into concrete actions, service robots are not meant for one-time use. Rather, they are meant to serve their function over the long-term, and any actions they take should be grounded in a *dynamic*, real-world environment. Specifically, this environment can be changed both by the actions of the service robot itself as well as by external influences like humans.

Let us decompose this larger goal by ignoring external changes to the environment and considering how an LLM-based agent can generate plans that are consistent with changes to the environment from past actions. One naive approach might be to, using an existing framework, generate a report of all past actions and feed this report alongside any prompts to the LLM so that the LLM is always aware of the most recent state of the environment. However, as the context from past actions grows, this approach is prone to model hallucination, which can significantly affect the accuracy of the agent [3]. This suggests that the LLM-based agent must have an external memory.

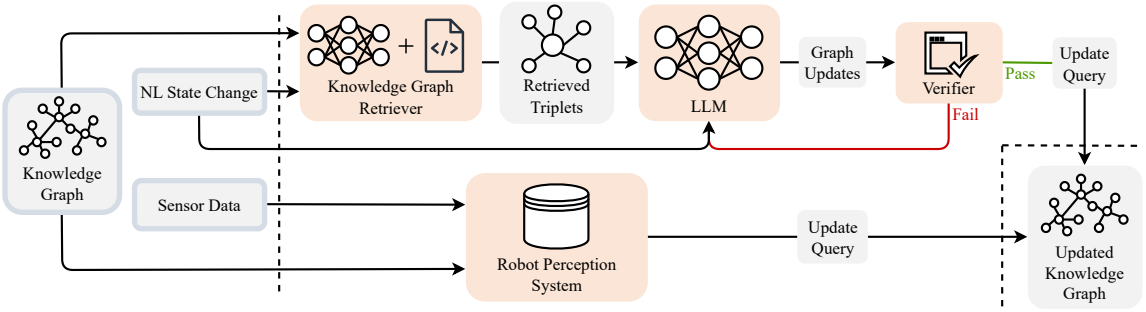
Now let us consider how an LLM-based agent can generate plans that are consistent with solely external changes to the environment. In many service robot applications, information on updates to the environment can arrive in many forms, including sensor input as well as dialogue with humans in the environment, where a human might describe a change in the environment. While there are already solutions for traditional knowledge representation in service robots that can be updated based on sensor input [4], [5], [6], updates based on human dialogue would require a natural language interface, probably using an LLM. This suggests that an LLM-based agent would need a knowledge base that is precise enough to be used within a traditional knowledge representation system but also simple enough to be used by an LLM to process natural language environment updates.

This motivation leads us develop a framework for augmenting existing research on grounded LLM-based agents with a dynamic memory that is both rigorously defined but also easily interpretable. We hope to design a system that can interface with this memory as follows:

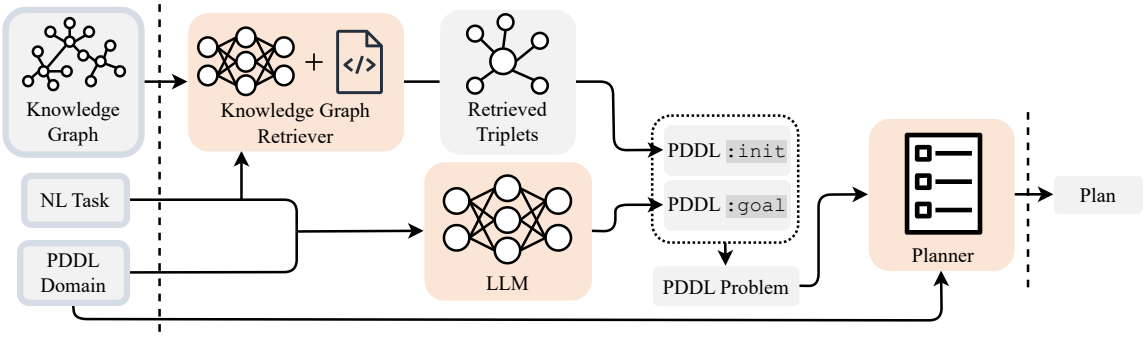
1) Introduce an LLM-based natural language interface for updating the memory given natural language descriptions of environment updates.
2) Integrate with robot perception so the memory can be updated based on sensory input.
3) Retrieve relevant information from the memory that can be used as context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

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

[§]Sony AI



(a) Update



(b) Plan

Fig. 1: (a) L3M+P maintains consistency of its knowledge graph through sensory input as well as descriptions of changes to the environment supplied by humans. Both sources are used to update the knowledge graph to reflect an up-to-date representation of the environment. (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.

II. BACKGROUND

A. Planning with Language Models

LLM+P [7] is an existing framework for combining LLMs with classical planning in robot agents to bridge the gap between natural language task descriptions and symbolic planners. These symbolic planners operate using the planning domain definition language (PDDL), a commonly used language for formally defining environments and tasks [8], [9]. A symbolic planner requires a domain PDDL file that defines a state-action space through (a) a set of predicates that will fully represent a state and (b) a set of actions for manipulating the current state. The domain file also defines the preconditions and effects of each action. To generate a plan for a specific task, a symbolic planner also requires a problem PDDL file that 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 initial state, a description of a task, and a domain PDDL 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.

We aim to extend LLM+P by removing the requirement for a description of the initial state and instead maintaining the current state in an internal knowledge base that can be used to extract information relevant to a given planning task. Using the relevant context, we can use LLM+P to generate a problem PDDL file that we can supply to a classical planner in order to produce a plan.

B. Knowledge Graphs

Knowledge graphs [10] are representations of knowledge 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. Some common knowledge graph technologies include Neo4j [11], one of the most widely adopted graph database systems, and Apache AGE [12], a PostgreSQL extension that allows integrating relational and graph databases. Similar to traditional databases, graph databases also generally support structured querying, such as through the Cypher query language [13].

We aim to use a knowledge graph as the internal knowledge base for L3M+P, as a knowledge graph provides a natural way to represent the world state and extract/insert information.

C. Retrieval-Augmented Generation

Many applications of LLMs involve using these models to answer domain-specific queries. One approach for adapting a pretrained model to this purpose could be to fine-tune the

model on domain-specific knowledge [14]. 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 [15]. 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. Some common RAG implementations use pretrained transformers to generate document embeddings and store the documents in a vector index [16]. When given a prompt, an embedding for the prompt is generated and used to perform a top-k retrieval of the documents, whose content can be added to the original prompt.

Our framework has a RAG component which retrieves the relevant edges from a knowledge graph and feed them as context for the LLM to handle state changes and plan queries.

III. PROBLEM STATEMENT

The overarching goal of our framework is to be able to solve two distinct types of problems: update and plan.

A. Update

The first problem is for the system to somehow register updates that take place in the environment. An update is provided either as natural language or as sensory input, and it could represent a single event, multiple events, or even a partial event (if the full details of an event are unknown). In our work, we focus on updates provided as natural language.

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 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.

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 update u_t. As a more simplified problem, a valid output is $\langle E_t^-, E_t^+ \rangle$, where we interpret $E_{t+1} = E_t - E_t^- + E_t^+$, so $s_{t+1} = \langle V_t, E_{t+1} \rangle$ (we assume entities are not added/removed from the environment).

B. Plan

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 robot knowledge at time t.
- g_t is a natural language 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.

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

A. Knowledge Graph Retrieval

To represent the state of the environment, we employ a knowledge graph 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. We may refer to the relationship between a pair of nodes as a triplet. These triplets can be written in one of two forms:

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

Each triplet in the first form can be interpreted as: the subject has the given relationship to the given object. For example, `(table, in_room, living_room)` indicates that the table is located in the living room. Each triplet in the second form can be interpreted as: it is true/false that the subject has the given property. For example, `(light, on, true)` indicates that the light is on.

These triplets have a one-to-one correspondence with PDDL predicates specified by the domain file. The knowledge graph is therefore a full representation of the current state $s_t \in \mathcal{S}$. It is important to note that we are assuming that the system is provided a domain PDDL file that specifies predicates representing the state of the environment as well as 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 knowledge graph. 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. Then, given a list of relevant entities, we extract all triplets outgoing and incoming to each relevant entity at a certain maximum depth. These retrieved triplets function as the context for the downstream task.

In case of wrong or undescriptive entity labels produced by the robot's object recognition system, the LLM may not be able to select the correct entities based on their names in the knowledge graph. We further propose a search-based retrieval method to precisely locate entities that match the natural-language description. We first prompt the LLM to generate a query graph with entities and their inferred relations from the natural-language query, and then search for the most similar grounded sub-graph in the knowledge graph. The sub-graph matching algorithm is inspired by recent works on scene graphs that support querying an instance in natural language [17], [18], but generalizes to finding all entities that are relevant to an update or task.

B. Updates to Knowledge Graph

A key contribution that L3M+P makes is the ability to dynamically update the agent's memory base based on external changes to the environment (shown in Fig. 1a). The agent can be alerted of these changes in two manners: (1) the robot receives sensory input or (2) a human in the environment describes an event in the environment to the agent in natural language. For our purposes, we will assume that the robot's sensory inputs are fully functional and can convert information directly into triplets to update in the knowledge graph through a traditional knowledge representation system. As such, we will focus on natural language descriptions of updates to the environment.

Any descriptions of environment changes will need to be reflected in the knowledge graph to maintain a consistent memory base. Given such a description, the following procedure is used:

1) Use the state change description to retrieve relevant triplets from the knowledge graph. Formally, the knowledge graph represents the current state $s_t = \langle V_t, E_t \rangle$, so partition $E_t = E_t^{rel} \cup E_t^{irrel}$ (where $E_t^{rel} \cap E_t^{irrel} = \emptyset$).
2) Provide the LLM with the entites V_t and the selected E_t^{rel} as well as the original event description. Prompt it to generate a set of triplets to remove from the graph E_t^- as well as a set of triplets to add to the graph E_t^+.
3) Perform a verification step to ensure that the generated E_t^- and E_t^+ are consistent with the domain PDDL. Specifically, check that the generated triplets are defined as PDDL predicates and the arguments in the triplet match in number and type with the predicate specification (a property triplet is only defined to have one PDDL predicate argument, for example).
4) If E_t^- and E_t^+ are consistent, update the graph as $E_{t+1}^{rel} = E_t^{rel} - E_t^- + E_t^+$, $E_{t+1} = E_{t+1}^{rel} + E_t^{irrel}$. Otherwise, re-prompt the LLM for update triplets, reporting the issues with the existing generation.

C. Planning

We make use of a modified version of the existing LLM+P framework to perform planning tasks (shown in Fig. 1b). Specifically, we do not have a user-provided description of the environment and instead query the knowledge graph to gain sufficient detail for a given planning task. When presented with a natural language description of a task, the following procedure is used:

1) Use the task description to retrieve relevant triplets from the knowledge graph. Formally, choose a relevant subset $s_t^{rel} \subseteq s_t$ of the current state s_t.
2) Fill in the `:init` block of a problem PDDL file with the retrieved s_t^{rel}.
3) Query the LLM to fill in the `:goal` block of the problem PDDL given the task description and the domain specification.
4) Pass the fully-constructed problem PDDL file to a symbolic planner alongside the provided domain PDDL file to generate a plan.
5) If the planner succeeds in finding a plan $\pi = \langle a_1, \ldots, a_N \rangle$, execute the specified actions a_i in order and concurrently update the knowledge graph state as $s_{t,i} \leftarrow f(s_{t,i-1}, a_i)$, where $s_{t,0} = s_t$ and $s_{t,N} = s_{t+1}$. Otherwise, provide the LLM with the planner output and iteratively re-prompt it to generate the `:goal` block.

V. EXPERIMENTS

Our experiments are designed to address the following questions:

1) Does RAG enable more accurate knowledge graph updates by allowing 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 knowledge graph state translate to correct plans, and conversely does incorrect knowledge graph state translate to failed plans? **(Yes)**

To answer these questions, we run two types of experiments. The first is meant to test solely text-based state changes and how L3M+P is able to both update the knowledge graph based on natural language state updates as well as generate plans based on a continuously updating knowledge graph. The second demonstrates that the framework seamlessly registers both verbal and sensory updates to the knowledge graph so that a service robot can robustly solve tasks in a dynamic environment.

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 a natural language description of each environment update and a natural language description of each planning task. The agent is also provided a domain PDDL file, which again is reasonable even in realistic use cases. To measure accuracy, the simulation maintains a ground-truth knowledge graph after each state update and plan execution to compare against L3M+P's proposed knowledge graph, 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.

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, and provide your response in JSON format.

GPT-4o (generated knowledge graph update):
```
{"REMOVE": ["laundry_room -> has ->
laundry_room_light"], "ADD": []}
```

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", ...]
```

`:goal` Block Prompt: Task + Provide me with the problem PDDL file that describes the planning problem directly without further explanations.

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)
...
```

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

1) R^-: The agent receives the full knowledge graph as context when performing state updates (no RAG).
2) R_V^-: The agent receives the full knowledge graph 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 knowledge graph that will fit 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: Same as R^+ but with search-based RAG.
6) R_V^S: Same as R_V^+ but with search-based RAG.

To execute these experiments, we use OpenAI's GPT-4o language model. We also utilize the SIW-THEN-BFSF planner provided by LAPKT [19] for all experiments. The goal of these experiments is not to prove the planning accuracy of LLM+P but rather to measure the effect of a potentially incorrect knowledge graph on generating correct plans.

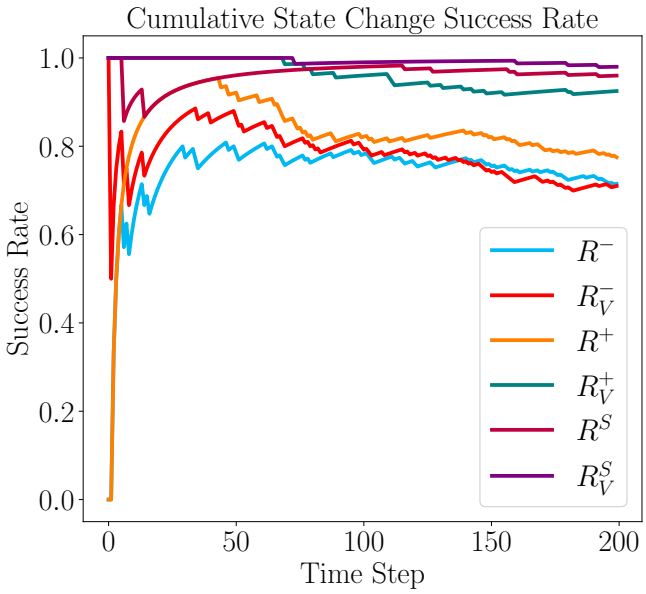


Fig. 2: Cumulative success rate of updating the knowledge graph from given state changes at each state change for all six variants of the agent.

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

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

Here are our findings:

1) As shown by the results in Table I, RAG helps prevent model hallucinations to significantly increase update accuracy, and it certainly does not worsen performance by not providing the full KG as context.
2) Also shown by Table I, syntactically verifying state changes generated by the LLM against the domain PDDL can also improve the accuracy of the state changes. While the knowledge graph context is able to generally inform the LLM of the proper syntax of the knowledge graph triplets, syntax verification serves as a stronger guarantee of correctness.
3) Other than some initial noise, Figure 2 demonstrates that each variant of the agent maintains a roughly consistent state change success rate throughout the run of the simulation, meaning the reported results are consistent rather than the result of luck.

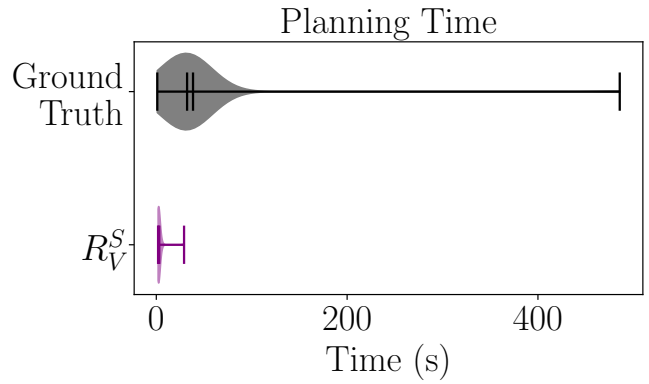


Fig. 3: Distributions of planner time for R_V^S (including time for RAG) and planner time against a ground truth PDDL with the full state provided, for tasks successfully solved by R_V^S.

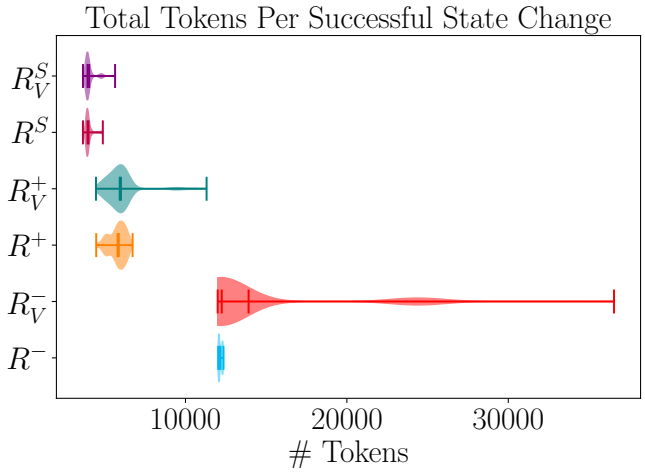


Fig. 4: Distributions of number of tokens used per successful state change for all six variants of the agent.

4) As expected, knowledge graph accuracy is analogous to plan-generation accuracy. In most failure instances, the LLM-generated PPDL `:goal` block is accurate, but the planner finds an inaccurate plan or fails to even find a plan as a result of an inaccurate knowledge graph state (which forms the `:init` block of the PDDL problem). This leads to plans that (a) only partially solve the task, or (b) plans that are unable to be executed due to their inconsistency with the true environment.
5) We also experience a time savings by using RAG: As shown in Figure 3, we have a speedup when comparing (a) the time taken by the planner when provided the full KG context as the `:init` block (ground truth), against (b) the total time to perform RAG and plan using retrieved triplets as a smaller context for the `:init` block (using R_V^S). For correct plans generated by R_V^S, this is an average speedup of 12.5x.
6) Lastly, we have a cost savings with RAG: As shown in Figure 4, the agent variants that use RAG use fewer tokens for processing successful state changes compared to the variants that do not use RAG. On average, we use 67.6% fewer total input and output tokens with R_V^S compared to R^- for processing natural language state changes (including tokens used for performing RAG).

B. Embodied Interactive Simulation

A service robot must be able to take recent changes into account while completing long-horizon tasks. For instance, the robot may be executing a plan for serving a cup of coffee by first grasping a clean cup from the cupboard, but then discovers or be informed that all cups are dirty. To re-plan for this scenario, the planning knowledge must be updated with execution-time information.

The L3M+P framework is instantiated in an embodied AI system in the AI2-THOR [20] simulator and tested on a variety of home tasks. We demonstrate that the L3M+P framework supports multi-modal state updates, which is essential for realistic home environments. At each time step, the agent is provided with the semantic properties of the currently visible objects in the simulation.

VI. Related Work

Several frameworks have already been proposed for LLM-based planning and reasoning [21], [22], [23], [24], [25]. However, by using an LLM as a plan generator, these frameworks make a critical error by mistaking language modeling capabilities for 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.

However, there are some frameworks that have been introduced to address planning within dynamic environments. LLM-DP [26] is a system that maintains a set of known information about the environment as well as a set of beliefs about the environment. These are gradually updated as the agent makes observations, and this information is then used as context for an LLM to generate a world specification and goal in PDDL that is provided to a classical planner. However, aside from the fact that this system was exclusively tested in Alfworld [27], 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 [28] is framework for maintaining and updating a world state to serve as a memory for an LLM-based agent across planning tasks. The current state s is represented in a JSON-like format, and the framework consists of both a world-state reader and a world-state writer. The world-state reader will provide s with a user planning query to an LLM and prompt it to generate a set of actions to achieve the goal. The world-state writer will feed the current state s and the generated plan into the LLM and prompt it to update the s from 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, in which case the world-state representation itself can exceed context-window limits of LLMs or at least significantly increase the likelihood of hallucinations.

In order to address the limitations of these other frameworks, we propose L3M+P, a system that will maintain a dynamic memory base for the agent in that can be updated and queried. Context from this memory base can be extracted at planning time to write accurate specifications about the environment and task that can be used by classical planners. This memory base is meant to change based on various, multi-modal sources of information about the environment, and it remains in a consistent format that can be used for writing planning specifications. The use of an external memory base will enable representing the absolute world state of fairly complex environments in a domain-agnostic manner, and the use of context retrieval will prevent issues with limited context windows.

VII. CONCLUSION

This paper introduces L3M+P, a novel framework that integrates LLMs with classical planning through a dynamic knowledge graph, addressing key limitations in reasoning, memory, and planning for long-term autonomous agents. By leveraging RAG and syntactic verification, L3M+P ensures consistency and accuracy in updating and querying knowledge graphs, enabling robust task execution in dynamic environments. Our experiments demonstrate significant improvements in both knowledge graph updates and planning accuracy, underscoring the framework's potential to enhance real-world applications like service robots. Future work could explore scaling this approach to more complex environments and integrating additional modalities for enriched perception and interaction. With these advancements, L3M+P stands as a promising step toward creating intelligent, context-aware robotic systems capable of lifelong optimal planning.

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] 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.

[4] 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

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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

[11] J. J. Miller, "Graph Database Applications and Concepts with Neo4j," in *Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA March 23rd-24th, 2013*. AIS eLibrary, 2013, pp. 141–147. [Online]. Available: https://aisel.aisnet.org/sais2013/24

[12] A. A. P. M. Committee, "Apache age's documentation," https://age.apache.org/age-manual/master/index.html, 2020–2024.

[13] 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.

[14] 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.

[15] 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.

[16] Z. Jing, Y. Su, Y. Han, B. Yuan, H. Xu, C. Liu, K. Chen, and M. Zhang, "When large language models meet vector databases: A survey," 2024.

[17] 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.

[18] 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.

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

[20] "AI2-THOR: An Interactive 3D Environment for Visual AI," *ArXiv*, vol. abs/1712.05474, 2017.

[21] 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.

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

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

[24] 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.

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

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

[27] 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

[28] 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.