AdaptBot: Combining LLM with Knowledge Graphs and Human Input for Generic-to-Specific Task Decomposition and Knowledge Refinement

Shivam Singh[1], Karthik Swaminathan[1], Nabanita Dash[1], Ramandeep Singh[1]
Snehasis Banerjee[2], Mohan Sridharan[3], Madhava Krishna[1]

***Abstract*— Embodied agents assisting humans are often asked to complete a new task in a new scenario. An agent preparing a particular dish in the kitchen based on a known recipe may be asked to prepare a new dish or to perform cleaning tasks in the storeroom. There may not be sufficient resources, e.g., time or labeled examples, to train the agent for these new situations. Large Language Models (LLMs) trained on considerable knowledge across many domains are able to predict a sequence of abstract actions for such new tasks and scenarios, although it may not be possible for the agent to execute this action sequence due to task-, agent-, or domain-specific constraints. Our framework addresses these challenges by leveraging the generic predictions provided by LLM and the prior domain-specific knowledge encoded in a Knowledge Graph (KG), enabling an agent to quickly adapt to new tasks and scenarios. The robot also solicits and uses human input as needed to refine its existing knowledge. Based on experimental evaluation over cooking and cleaning tasks in simulation domains, we demonstrate that the interplay between LLM, KG, and human input leads to substantial performance gains compared with just using the LLM output.**
Project website[§]: https://sssshivvvv.github.io/adaptbot/

***Index Terms*— Large Language Models, Knowledge Graph, Human-in-the-loop Learning**

1 Introduction

Embodied agents are being used in assistive roles in many applications, aided in part by the availability of realistic simulators [1]–[3]. Although such agents possess some prior knowledge of domain objects and their attributes, they are often asked to perform new tasks and operate in new scenarios. For example, an agent preparing dishes in the kitchen based on prior knowledge of some recipes and ingredients, may be asked to prepare a new dish or clean the pantry.

Large Language Models (LLMs) trained on a large corpus of data have demonstrated the ability to decompose a range of tasks into a sequence of high-level (abstract) actions (i.e., sub-tasks) that implement the task [4]–[6]. For example, an LLM can provide a sequence of sub-tasks for completing the previously unseen task of *preparing hot chocolate*. However, this sequence may involve incorrect steps, or reference objects and actions that the agent does not have access to in the kitchen under consideration.

The challenges mentioned above are partially offset by the fact that an assistive agent usually has some prior domain-specific knowledge in the form of objects, object attributes, and action capabilities. State-of-the-art methods build large datasets of such information for a given application domain [7], or attempt to embed this knowledge by repeatedly tuning deep networks [8]. However, such knowledge is not readily available for many practical domains, and modern data-driven methods make it difficult to reliably and transparently revise the encoded knowledge over time. In a departure from such methods, the framework described in this paper seeks to leverage the complementary strengths of LLMs, Knowledge Graphs (KGs), and human feedback—see Figure 1. Our framework enables the assistive agent to:

1) Query an LLM to obtain a generic sequence of actions (sub-tasks) to be executed to accomplish any given task.
2) Encode any prior domain-specific knowledge of object types and attributes in a KG, using it to revise the LLM's output action sequence.
3) Use discrepancies between LLM output, KG, and ob-

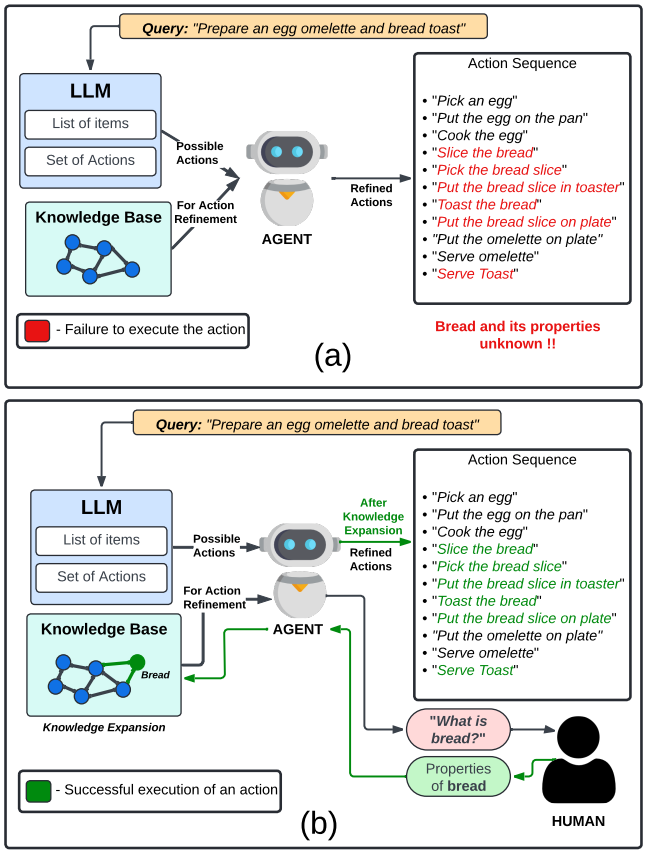

Fig. 1: For any given task, an LLM provides a generic sequence of abstract actions that is refined using the domain-specific knowledge in a KG. If the sequence refers to objects, attributes, or actions that cannot be resolved using the KG, or leads to unexpected outcomes, human input helps refine or expand the KG.

[1] Robotics Research Center, IIIT Hyderabad, India
[2] TCS Research, Tata Consultancy Services, India
[3] School of Informatics, University of Edinburgh, UK
[§]Project supported in part by TCS Research India

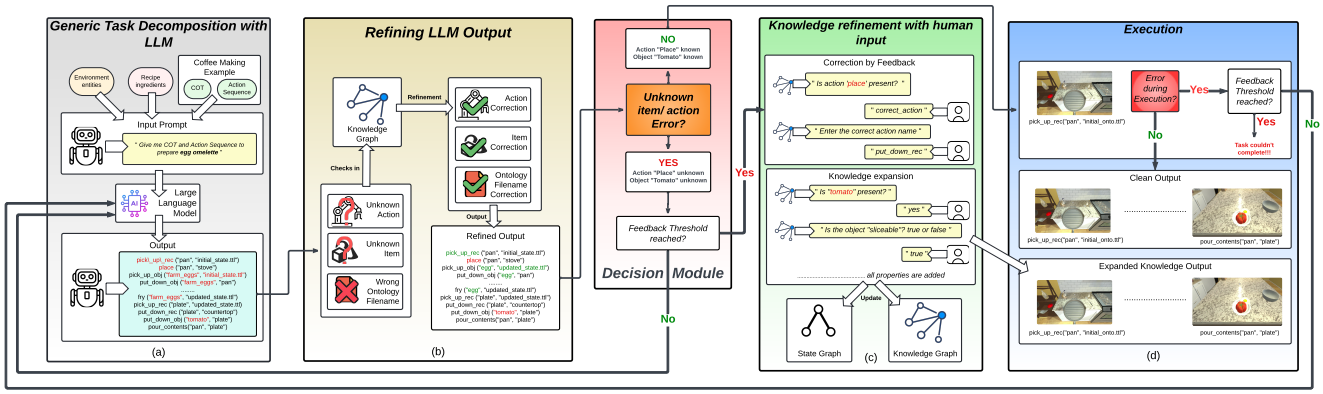

Fig. 2: Framework overview for cooking tasks: (a) Input Chain-of-Thought (COT) prompt contains target dish, available ingredients, and an example of input and output action sequence (for task of making coffee), to obtain an output action sequence; (b) Any mismatch (e.g., in object classes, actions) between LLM output and KG are identified and action sequence is revised if possible; (c) Agent attempts to resolve any remaining errors or unexpected outcomes by re-prompting LLM, with errors that persist being addressed by soliciting human input and updating KG; (iv) Revised/corrected action sequence is executed.

servations of action outcomes to support human-in-the-loop (HITL) refinement of the knowledge in the KG.

We illustrate and evaluate these capabilities in two different classes of tasks: cooking and cleaning, demonstrating: (a) substantial improvement in performance compared with baselines that use just the LLM or even a combination of LLM and KG for completing an assigned task; and (b) the ability to adapt to new classes of tasks through incremental knowledge refinement instead of elaborate tuning (e.g., of LLMs) or encoding comprehensive knowledge.

The remainder of the paper is organized as follows. We begin with a discussion of related work (Section 2), followed by a description of our proposed framework (Section 3). We then discuss the experimental set up and results (Section 4), followed by the conclusions (Section 5).

2 Related Work

We motivate our framework by discussing related work in the use of LLMs, KGs, and HITL task decomposition.

LLMs and KGs for task decomposition: LLMs such as GPT-4 [9], Gemma2 [10], and LLaMA3 [11] have experimentally demonstrated the ability to decompose abstract tasks into sub-tasks [4]–[6], [12]–[14]. Frameworks such as TaskBench [15] have compared fully automated processes with those with human interventions, particularly for unfamiliar or "open-set" tasks [6], [16], [17]. Additionally, methods such as ADaPT [18] have supported iterative adjustment of task complexity continuously based on real-time feedback. In parallel, KGs have have been used to model prior knowledge of objects and their attributes for sequential task planning, e.g., for sequential task prediction with graph CNN [19], action planning for robots in Industry 4.0 environments [20], and for generalizing to new (related) environments [21]. Our framework builds on these ideas by combining the (generic) prediction capabilities of LLMs with the domain-specific knowledge encoded in a KG [22], [23] for task adaptation in new environments [24].

Related task planning examples: The Functional Object-Oriented Network (FOON) [25] encodes substantial knowledge about cooking (e.g., ingredients and outcomes of actions) in the form of task trees and using them for task planning for cooking related dishes [7], [26]–[28]. In more recent work, a fine-tuned GPT has been used to transform generic recipe instructions into task trees, which are merged and revised by comparing information stored in FOONs to obtain the task tree used for execution [8]. These methods use examples from the Recipe1M+ dataset [29] for tuning and evaluation. Instead of tuning an LLM across classes of tasks or training a knowledge base extensively for a particular class of tasks, our framework supports incremental revision, faster adaptation, and reliability. Our framework provides the assistive agent limited (prior) knowledge of any specific domain as a KG, enabling it to incrementally refine the KG with new objects and actions as they are encountered, and to correct errors by soliciting and using human feedback when it is necessary and available.

Human-in-the-loop task decomposition: Human feedback has been used to enhance hierarchical task allocation and robot task planning in complex environments [30], [31]. Frameworks like TaskBench [15] and Reflexion [32] leverage human feedback to iteratively decompose tasks, making LLMs more effective in handling abstract tasks. Hierarchical task structuring is crucial for handling complex, multi-step task decomposition, especially in abstract problem domains [33]. Instead of iteratively tuning LLMs (e.g., through prompts), which does not necessarily lead to correct results, we combine the generic prediction capabilities of LLM, real-time domain-specific KG updates [34], [35], and human-in-the-loop feedback [36]–[40], allowing the system to operate based on the available knowledge to perform new classes of tasks while incrementally refining the knowledge.

3 Problem Formulation and Framework

Figure 2 is an outline of our framework. In the motivating example, an agent assisting in cooking tasks in a kitchen has access to relevant objects and ingredients for many dishes but it does not have the recipes. When asked to prepare any particular dish, τ_i, the agent queries an LLM to obtain a sequence of abstract actions (sub-tasks), i.e., $\langle a_1, \ldots, a_{m_i} \rangle$. For example, the sequence for *make an omelette* includes

picking up the egg and *breaking the egg over a skillet*. This sequence of abstract actions is checked against a KG with some domain-specific information in the form of existing objects and attributes that include the actions that can be performed on some objects. The agent tries to resolve any discrepancy between the LLM output and KG, e.g., KG states there is no skillet or that an egg can only be cracked, by finding replacements, e.g., *crack the egg over a pan*. If the discrepancy is not resolved, or if executing the action sequence does not provide the desired outcome, the agent identifies relevant actions and solicits human input to refine the KG, e.g., add knowledge of objects or their attributes, and provides an action sequence to complete the task. The agent is assumed to be able to execute these actions. We describe out framework's components below.

3.1 Generic Task Decomposition with LLM

In our framework, we use an LLM to decompose any given task into a sequence of sub-tasks because LLMs have demonstrated the ability to provide such a sequence of abstract actions for many different tasks. Specifically, in the motivating example, the LLM is prompted with information about some domain objects, an example cooking task (make coffee), and the corresponding action sequence (recipe) to be executed—see Figure 2a. We experimentally evaluate the use of different LLMs, as described in Section 4.1.

Since the sequence of sub-tasks predicted by the LLM is based on many information sources, it may not be possible to execute one or more of these actions. For example, in the context of cooking tasks, the suggested ingredient may not be available or the action may involve an incorrect choice of tool (e.g., using a fork to cut vegetables). These situations can be addressed in part by using prior domain-specific information, which is encoded as described below.

3.2 Representing Domain-specific Knowledge with KG

Our framework uses a Knowledge Graph (KG) to encode any prior information available to the agent. In the context of cooking tasks, this includes knowledge of some classes of ingredients (e.g., herbs, fruits, vegetables), receptacles (e.g., plates, bowls, countertop), and tools (e.g. knives, spoons), which can be arranged hierarchically. It also encodes the existence of some specific instances of these object classes and their properties such as likely location(s) and the actions they can be involved in (e.g., cutting, scooping, grinding). We use the *Resource Description Framework* (RDF) format to encode this information in two graph structures in Turtle format (.ttl file)—see Figure 3:

1) **State graph:** models current state as $\mathbf{G_s} = (\mathbf{I_s}, \mathbf{E_s})$, where nodes $\mathbf{I_s}$ are instances of object classes such as ingredients and receptacles; and $\mathbf{E_s} \subseteq \mathbf{I_s} \times \mathbf{P_s} \times \mathbf{V_s}$ are edges such that $(i_j, p, v_k) \in \mathbf{E_s}$ is a triple denoting an attribute of $i_j \in \mathbf{I_s}$ in terms of value $v_k \in \mathbf{V_s}$ of predicate $p \in \mathbf{P_s}$. For example, (apple1, obj_location, fridge) and (apple1, is_sliced, true) express *apple1's* location and that it is sliced.

```
ex:onion rdf:type ex:object ;
    ex:obj_name 'onion' ;
    ex:IsSliceable true ;
    ex:Fryable true ;
    ex:NeedsToBeCleaned true .
```

```
ex:onion rdf:type ex:object ;
    ex:obj_name 'onion' ;
    ex:obj_location ex:fridge .
    ex:sliced false ;
    ex:IsFried false ;
    ex:IsCleaned false .
```

Fig. 3: Example of a node *onion* in $\mathbf{G_k}$ (*top*) and $\mathbf{G_s}$ (*bottom*).

2) **Attribute graph:** encodes the known properties and action capabilities of some object classes as $\mathbf{G_k} = (\mathbf{I_k}, \mathbf{E_k})$, where nodes $\mathbf{I_k}$ represent the classes and edges $\mathbf{E_k} \subseteq \mathbf{I_k} \times \mathbf{P_k} \times \mathbf{V_k}$ represent class properties, e.g., (apple, sliceable, true) implies apples can be sliced.

The available actions include moving, picking up, and putting down objects; using tools; cleaning, toggling, slicing, stirring, and mopping*. Such a KG can be learned automatically based on information extracted from datasets or sensor streams. The feasibility of any action/sub-task in the sequence predicted by LLM is then checked using $\mathbf{G_k}$ and $\mathbf{G_s}$ by generating suitable SPARQL queries. If the predicted sequence of actions passes the KG-based check, it is executed, changing $\mathbf{G_s}$ suitably.

3.3 Refining LLM output

If a mismatch is detected between the LLM output and the KG, the agent attempts to use the KG to *revise* the action sequence—see Figure 2(b). Specifically, the agent attempts to replace the text corresponding to the identified mismatch, which can refer to actions, object instances, or object attributes, with other text from the KG. While performing such text replacement, it is important to consider syntactic similarity, which measures similarity in the structure (e.g., of words or sentences), and semantic similarity, which considers similarity in meaning. In our framework, the agent can compute the similarity of the identified words (or their embedding) with words from a similar category (or their embedding) in the KG. The use of word embeddings requires additional contextual information and makes it difficult to understand the revision of the LLM output. We thus chose to use the direct matching of words while considering hypernyms (broader terms) or hyponyms (more specific terms) for simplicity, ease of use, and transparency. If the agent is able to replace all identified mismatches, it executes the actions.

3.4 Knowledge refinement with human input

Since the KG is not comprehensive, the agent may not be able to resolve all identified mismatches, e.g., reference to unknown object or action. Also, there may be unexpected action outcomes when the agent executes the action sequence. These situations are handled through re-prompting and human feedback—see Figure 2(c). Specifically, the agent responds to an unresolved mismatch or erroneous outcome

*Supplementary material includes list of all the actions.

by re-prompting the LLM with additional information (of mismatch or error). If the mismatch or error persists, human input is solicited and used.

Existence check: if an action or object in the LLM output does not exist in the KG, there are three possibilities: (1) The agent is mistaking an existing item (action) for another item (action); (2) the entity does not exist in the domain; or (3) the entity exists but is not in the KG. In the first case, human informs the agent about the correct object (or action); in the second case, human denies existence of entity; and in the third case, human confirms the entity's existence and agent interactively obtains entity's attributes[†].

Learn attributes: If human confirms existence of an instance of a new entity, the agent interactively obtains additional details. For example, when informed about an instance of a new object class *onion*, agent incrementally requests information about the object type (e.g., *edible_object*) and other relevant attributes (e.g., boilable, fryable, location of instance). This knowledge revision can be viewed as correcting (expanding) the knowledge in the KG by revising class attributes in $\mathbf{G_k}$ and instance-specific details in $\mathbf{G_s}$.

$$f_{KE}(I_{new}, P_{new}, S_{current}) \Rightarrow \mathbf{G_k^{'}}, \mathbf{G_s^{'}}$$

where I_{new} is the new entity; $P_{new} = (p_1, v_1), ..., (p_n, v_n)$ refers to attributes (p_i) of entity and their values (v_i); $S_{current} = (s_1, v_1), ..., (s_n, s_n)$ refers to states s_i and their values v_i; and $\mathbf{G_k^{'}}$ and $\mathbf{G_s^{'}}$ are the updated components of the KG. For example, new edge is added in $\mathbf{G_s}$ to encode an onion's position and new edge is added in $\mathbf{G_k}$ to encode that an onion can be fried. Note that this update to existing knowledge is fully transparent by design.

Algorithm 1 describes the flow of information and control in our framework. The framework takes as input the state graph $\mathbf{G_s}$ and attribute graph $\mathbf{G_k}$, along with an input prompt *(ip_prompt)* that contains information about the class of tasks, an in-context example, and a query specifying the task the agent must perform. The LLM generates an action sequence T (Line 3), which is refined to $T_{refined}$ using the knowledge in the KG (Line 4). If there are no unresolved mismatches between KG and LLM output (ε_{unkn}), the action sequence is executed, with the outcomes and errors collected for further analysis (Lines 5-7). Any unresolved mismatches or errors in outcome result in a feedback prompt to the LLM, leading to a new predicted sequence of actions $T^{'}$ (Lines 9-13, 19-23). If these mismatches and/or errors persist (beyond threshold F_{max}), the agent queries a human, which potentially leads to knowledge refinement, updating $\mathbf{G_k}$ and $\mathbf{G_s}$. After the expansion, the knowledge base is updated, and the refined action sequence is executed and evaluated (Lines 14-18, 24-26). This entire process is repeated until the tasks is completed or some threshold (e.g., time limit) is exceeded.

4 Experimental Sections and Results

This section describes the experimental setup and the results of experimentally evaluating three hypotheses:

[†]Supplementary material includes details of questions asked.

Algorithm 1 LLM + KG + Human Input

1: **Procedure** LLM_KG_Human($\mathbf{G_s}$, $\mathbf{G_k}$, *ip_prompt*)
2: $F \leftarrow 0$ ▷ F is feedback counter
3: $T \leftarrow$ call_LLM(*ip_prompt*) ▷ T is action sequence
4: $T_{refined}$, $\varepsilon_{unkn} \leftarrow$ refine_sequence(T, $\mathbf{G_k}$, $\mathbf{G_s}$)
5: **if** NOT ε_{unkn} **then** ▷ ε_{unkn} is unknown_item error
6: O, $\varepsilon_{exec} \leftarrow$ execute($T_{refined}$) ▷ O is execution output ▷ ε_{exec} is execution error
7: **end if**
8: **while** (ε_{exec} OR ε_{unkn}) AND $F < F_{max}$ **do**
9: **while** ε_{unkn} AND $F < F_{max}$ **do**
10: $T^{'} \leftarrow$ call_LLM(fb_prompt) ▷ $T^{'}$ is updated sequence
11: $T^{'}_{refined}$, $\varepsilon_{unkn} \leftarrow$ refine_sequence($T^{'}$, $\mathbf{G_k}$, $\mathbf{G_s}$)
12: $F \leftarrow F + 1$
13: **end while**
14: **if** ε_{unkn} AND $F == F_{max}$ **then**
15: $T^{'}_{refined} \leftarrow$ ask_human(ε_{unkn}) ▷ $\mathbf{G_k}, \mathbf{G_s} \Rightarrow \mathbf{G_k^{'}}, \mathbf{G_s^{'}}$
16: O, $\varepsilon_{exec} \leftarrow$ execute($T^{'}_{refined}$)
17: **break**
18: **end if**
19: **if** ε_{exec} AND $F < F_{max}$ **then**
20: $T^{'} \leftarrow$ call_LLM(fb_prompt)
21: $T^{'}_{refined}$, $\varepsilon_{unkn} \leftarrow$ refine_sequence($T^{'}$, $\mathbf{G_k}$, $\mathbf{G_s}$)
22: $F \leftarrow F + 1$
23: **end if**
24: **if** NOT ε_{unkn} **then**
25: O, $\varepsilon_{exec} \leftarrow$ execute($T^{'}_{refined}$)
26: **end if**
27: **end while**
28: **End Procedure**

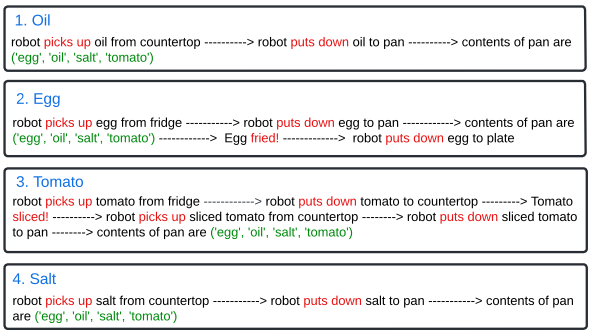

Fig. 4: Progress line [8] showing use of each ingredient when preparing an *omelette*.

H1: Combining generic prediction of action sequences from LLMs with KG-based specific prior knowledge improves performance compared with just LLMs.

H2: Soliciting and using human feedback as needed supports incremental knowledge revision and results in improved performance compared with not using human feedback.

H3: Our framework adapts to new classes of tasks through incremental and transparent knowledge refinement.

4.1 Experimental Setup

We begin by describing the experimental set up, which includes the prompting of LLMs, and the choice of baselines, classes of tasks, and evaluation measures.

```
'clean the bedroom_floor',
'dust the TV',
'wash the clothes',
'wash the dishes',
'water the plants',
'take out trash',
'clean the window',
'mop the countertop',
'clean the table',
'pick up and put all the toys in the toy box',
'charge the phone',
'play the music'
```

Fig. 5: 12 variants of tasks that involve the agent assisting with cleaning different objects and surfaces, or clearing objects to achieve the desired object configuration.

4.1.1 LLM Prompting: We used GPT-3.5 and GPT-4o to generate action sequences for specific tasks in a given environment. We used Chain of Thought (CoT) prompting to encourage the LLM to decompose any given task into a series of logical steps. The main prompt included domain-specific information (e.g., object classes from $\mathbf{G_s}$), set $\mathbf{A}$ of agent's actions, and output for a single in-context example. For example, for the agent assisting in cooking tasks, the LLM was encouraged to generate an action sequence that fetches ingredients and tools, completes the cooking process, and serves the dish, based on the example of preparing coffee. The LLM's output was filtered to retain only the predicted action sequence. As described in Section 3.4, any unresolved mismatch between LLM and KG, or an error in outcome, also led to the agent sending a feedback prompt to the LLM in an attempt to fix the error.

4.1.2 Baselines: We evaluated three different configurations of components in our framework: (a) LLM; (b) LLM with a KG; and (c) LLM with KG and human input (LLM + KG + Human). We conducted linked trials, i.e., in each trial, the same LLM output was provided to each configuration. As stated in Section 3, with just LLM, the predicted action sequence is sent directly for execution, and errors results in a feedback prompt to the LLM for a fixed number of times. With *LLM + KG*, the KG is used to identify and fix mismatches between LLM output and KG; however, consistent mismatches and incorrect execution outcomes are not addressed. The *LLM+ KG + human* configuration represents our framework, in which unresolved mismatches are addressed using human input, which is assumed to be accurate; the other two configurations serve as baselines.

4.1.3 Classes of tasks: In order to evaluate the ability of our framework to adapt to different classes of tasks, we considered cooking and cleaning tasks. Specifically, we considered 30 different cooking tasks in a kitchen; this is the motivating scenario described in Section 3. These tasks were created by sampling from the Recipe1M+ dataset [29]. In addition, we considered 12 variants of cleaning/clearing tasks that involved the agent cleaning specific objects or surfaces (e.g., "do the laundry"), or arranging objects in desired configurations in particular rooms (e.g., "clear the toys from the playroom")—see Figure 5 for some examples. The results of evaluating the adaptability of our framework is summarized later in Table II.

4.1.4 Evaluation Strategy: For the evaluation of our framework, we used human participants to provide ground truth. Specifically, we recruited 18 human evaluators to mark the execution outputs for each task assigned to the framework and the two baselines. The tasks were distributed such that the output for each task was evaluated by at least three human participants. The scores provided by the human (on a linear scale between 0-20) were averaged to obtain the success rate of our framework and the two baselines. These results are discussed further in Section 4.2.

To better understand the LLM's performance, we also considered *progress lines* [8], which depicted the use of key individual objects during individual steps of the action sequence, e.g., Figure 4 shows the movement of each ingredient when cooking an omelette. These were presented along with the execution outputs to be evaluated by the humans.

4.1.5 Evaluation Measures: The key performance measures considered in this work include:

- **Success rate:** As stated above, this measure was computed based on the scores assigned by the human participants. Higher values are better as they indicate a higher degree of satisfaction in task completion. This measure was used for evaluating H1-H3.
- **Average tokens used:** The number of tokens used when prompting the LLM (including the input prompt and all subsequent feedback prompts) was averaged across all tasks. This is a measure of resource consumption and lower values are usually better, except when the use of prompts improves the values of other measures.
- **Number of nodes and edges in KG:** We use this measure to evaluate H2 and H3. An increase in its value implies an expansion of knowledge in the KG.
- **Mean ingredient overlap:** A measure specific to the first class of tasks (cooking); it is the average overlap between the ingredients in the ground truth recipe and the ingredients in the executed action sequence. If m_i denotes the ingredients required to make a particular dish and l_i denotes the ingredients in the action sequence, this measure is computed as:

$$\text{Mean ingredient overlap} = \frac{1}{N}\sum_{i=1}^{N}\frac{|m_i \cap l_i|}{|m_i|} \quad (1)$$

where $|\cdot|$ is the cardinality of a set, and N is the total number of recipes sampled from the dataset. This measure was used to evaluate H1-H2.

4.2 Experimental Results

Next, we describe and discuss the experimental results.

Evaluating H1. We first explored whether the combination of LLM and KG leads to improved performance in comparison with just using LLM output for any given task. The corresponding results for the cooking-related tasks are summarized in Table I; in particular, see columns labeled "LLM" and "LLM + KG". For the two LLMs considered (GPT3.5, GPT4o), we observe a substantial increase in success rate, reduction in token use, and an increase in the mean ingredient overlap for LLM+KG compared with LLM. These results provide strong support for **H1**.

LLM Models ↓	Frameworks →	LLM	LLM + KG	LLM + KG + Human
GPT 4o	**Success Rate (in %)** ↑	45.2	56.95	**91.14**
	Avg. Tokens Used ↓	8316	7591	6459
	Mean Ingd. Overlap (in %)	56.7	65.27	**92.07**
	(#nodes, #edges) in $\mathbf{G_s}$ and $\mathbf{G_k}$	(79, 772)	(79, 772)	**(87, 845)**
GPT 3.5	**Success Rate (in %)** ↑	25.41	33.95	**92.08**
	Avg. Tokens Used ↓	8402	8415	4354
	Mean Ingd. Overlap (in %) ↑	38.13	44.29	**98.97**
	(#nodes, #edges) in $\mathbf{G_s}$ and $\mathbf{G_k}$	(79, 772)	(79, 772)	**(89, 869)**

TABLE I: Evaluating **H1 & H2** for 30 recipes of six categories from Recipe1M+ dataset. The combination of LLM and KG ("LLM+KG") results in an increase in success rate, reduction in token use, and an increase in the mean ingredient overlap compared with just using the LLM ("LLM"). Also, soliciting and using human input when needed ("LLM+KG+Human") results in a substantial improvement on all measures, including an increase in the number of nodes and edges due to expansion of knowledge in KG.

LLM Models ↓	Frameworks →	LLM	LLM + KG	LLM + KG + Human
GPT 4o	**Success Rate (in %)** ↑	42.33	44.00	**75.66**
	Avg. Tokens Used	3820	3571	1979
	(#nodes, #edges) in $\mathbf{G_s}$ and $\mathbf{G_k}$	(39, 313)	(39, 313)	**(40, 331)**
GPT 3.5	**Success Rate (in %)** ↑	32.63	42.5	**98.75**
	Avg. Tokens Used ↓	4963	4440	3510
	(#nodes, #edges) in $\mathbf{G_s}$ and $\mathbf{G_k}$	(39, 313)	(39, 313)	**(44, 397)**

TABLE II: Evaluating **H3** by adapting our framework to the cleaning and clearing tasks without requiring extensive tuning (e.g., of LLM) or comprehensive encoding of knowledge (in KG). We observed a substantial improvement on all performance measures with our framework compared with just using LLM outputs or even LLM+KG. Also, the agent is able to solicit human input as needed to incrementally and transparently revise knowledge in the KG.

Evaluating H2. Next, we explored the impact of soliciting and using human input as needed. The last column of Table I ("LLM+KG+Human") shows that our framework's judicious use of human input with LLM and KG markedly improved performance on all measures compared with LLM and LLM+KG. With GPT-4o, we observed a 45.94% increase in success rate over LLM and 34.19% over LLM+KG. For GPT-3.5, the success rate increased by 66.67% over LLM and 58.13% over LLM+KG. Also, the average number of tokens used by our framework dropped by 48.26% compared with baseline(s). This performance improvement was strongly influenced by the refinement of knowledge in the KG; the number of nodes and edges in the KG expanded from (79, 772) to (87, 845) with GPT-4o and to (89, 869) with GPT3.5. These results strongly support **H2**.

Evaluating H3. Finally, we evaluated the ability to adapt our framework to a different class of tasks (cleaning and clearing), with the results summarized in Table II. Unlike prior work [8], we seek to achieve this adaptation without extensive tuning (e.g., of LLM) or the need for comprehensive domain-specific knowledge (in the KG). We instead leverage the interplay between LLM, KG, and human input to support incremental adaptation to the new class of tasks. Results indicate (once again) a substantial improvement on all measures for our framework compared with the baselines. We noted that the impact of adding different bits of knowledge to the KG can differ. For example, with GPT-4o, the addition of just one item (mopping_cloth) to the KG based on human input led to a 31% increase in success rate; with GPT3.5, this improvement was more pronounced (56%). We also observed a substantial reduction in the number of tokens used. In addition, this adaptation of knowledge is fully transparent by design. These results strongly support hypothesis **H3**.

Our project web site[‡] hosts our supplementary material, including examples of tasks being performed in simulation, and supporting results with other LLM models.

[‡] https://sssshivvvv.github.io/adaptbot/

5 Conclusions and Future Work

Embodied agents assisting humans frequently have to complete previously unseen tasks or operate in new scenario. This paper describes a framework that leverages the complementary strengths of Large Language Models (LLMs), Knowledge Graphs (KGs), and Human-in-the-Loop (HITL) feedback to satisfy this requirement. Specifically, the generic task decomposition ability of LLMs is used to predict a sequence of abstract actions to complete any given task. This sequence is adapted to the specific scenario(s) and the task-, agent-, or domain-specific constraints using a KG that encodes prior knowledge of some objects, object attributes, and action capabilities. Any unresolved mismatch between the KG and the LLM output, and any unexpected action outcomes, are addressed by soliciting and using human input. This HITL feedback corrects errors and refines the existing knowledge (in the KG) for subsequent operation. Experimental evaluation in two simulated domains demonstrates substantial performance improvement compared with baselines, and illustrates incremental acquisition of knowledge to adapt to new classes of tasks.

This research opens up multiple avenues for further research. First, we will explore the use of this framework in many more classes of tasks, building on (and reinforcing) the promising results obtained so far. Second, we will investigate the trade-off between automating the generation of an action sequence for any given task, and soliciting and incorporating human feedback as needed. Furthermore, we will explore the use of this framework on a physical robot platform assisting humans. The long-term objective is to create assistive agents and robots that can interact and collaborate with humans in different application domains.

References

[1] E. Rohmer *et al.*, "Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework," in *Proc. of The Interna-*

tional Conference on Intelligent Robots and Systems (IROS), 2013, www.coppeliarobotics.com.

[2] X. Puig *et al.*, "Virtualhome: Simulating household activities via programs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[3] E. Kolve *et al.*, "Ai2-thor: An interactive 3d environment for visual ai," 2022. [Online]. Available: https://arxiv.org/abs/1712.05474

[4] T. Khot *et al.*, "Decomposed prompting: A modular approach for solving complex tasks," 2023. [Online]. Available: https://arxiv.org/abs/2210.02406

[5] J. Reppert *et al.*, "Iterated decomposition: Improving science q&a by supervising reasoning processes," 2023. [Online]. Available: https://arxiv.org/abs/2301.01751

[6] Y. Liu *et al.*, "Delta: Decomposed efficient long-term robot task planning using large language models," 2024. [Online]. Available: https://arxiv.org/abs/2404.03275

[7] M. S. Sakib *et al.*, "Approximate task tree retrieval in a knowledge network for robotic cooking," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 11 492–11 499, 2022.

[8] M. S. Sakib and Y. Sun, "From cooking recipes to robot task trees–improving planning correctness and task efficiency by leveraging llms with a knowledge network," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 12 704–12 711.

[9] OpenAI *et al.*, "Gpt-4 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2303.08774

[10] G. Team *et al.*, "Gemma 2: Improving open language models at a practical size," 2024. [Online]. Available: https://arxiv.org/abs/2408.00118

[11] Dubey *et al.*, "The llama 3 herd of models," 2024. [Online]. Available: https://arxiv.org/abs/2407.21783

[12] J. Wen *et al.*, "Learning task decomposition to assist humans in competitive programming," *arXiv preprint arXiv:2406.04604*, 2024.

[13] W. Li *et al.*, "Semantically aligned task decomposition in multi-agent reinforcement learning," *arXiv preprint arXiv:2305.10865*, 2023.

[14] L. M. Dery *et al.*, "Auxiliary task update decomposition: The good, the bad and the neutral," 2021. [Online]. Available: https://arxiv.org/abs/2108.11346

[15] Y. Shen *et al.*, "Taskbench: Benchmarking large language models for task automation," *arXiv preprint arXiv:2311.18760*, 2023.

[16] Y. Wang *et al.*, "Tdag: A multi-agent framework based on dynamic task decomposition and agent generation," *arXiv preprint arXiv:2402.10178*, 2024.

[17] G. Cui, W. Shuai, and X. Chen, "Semantic task planning for service robots in open worlds," *Future Internet*, vol. 13, no. 2, p. 49, 2021.

[18] A. Prasad *et al.*, "Adapt: As-needed decomposition and planning with language models," *arXiv preprint arXiv:2311.05772*, 2023.

[19] D. Zheng *et al.*, "A knowledge-based task planning approach for robot multi-task manipulation," *Complex & Intelligent Systems*, vol. 10, 07 2023.

[20] A. Kattepur and B. P, "Roboplanner: autonomous robotic action planning via knowledge graph queries," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, ser. SAC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 953âĂŞ956. [Online]. Available: https://doi.org/10.1145/3297280.3297568

[21] A. Daruna *et al.*, "Towards robust one-shot task execution using knowledge graph embeddings," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 11 118–11 124.

[22] H. Abu-Rasheed *et al.*, "Knowledge graphs as context sources for llm-based explanations of learning recommendations," 2024. [Online]. Available: https://arxiv.org/abs/2403.03008

[23] S. Pan *et al.*, "Unifying large language models and knowledge graphs: A roadmap," *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[24] Y. Kuang *et al.*, "Openfmnav: Towards open-set zero-shot object navigation via vision-language foundation models," 2024. [Online]. Available: https://arxiv.org/abs/2402.10670

[25] D. Paulius *et al.*, "Functional object-oriented network for manipulation learning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 2655–2662.

[26] Y. Ding *et al.*, "Robot task planning and situation handling in open worlds," 2022. [Online]. Available: https://arxiv.org/abs/2210.01287

[27] V. Bhat *et al.*, "Grounding llms for robot task planning using closed-loop state feedback," 2024. [Online]. Available: https://arxiv.org/abs/2402.08546

[28] Y.-q. Jiang *et al.*, "Task planning in robotics: an empirical comparison of pddl-and asp-based systems," *Frontiers of Information Technology & Electronic Engineering*, vol. 20, pp. 363–373, 2019.

[29] J. Marın *et al.*, "Recipe1m+: A dataset for learning cross-modal embeddings for cooking recipes and food images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 1, pp. 187–203, 2021.

[30] L. Marzari *et al.*, "Towards hierarchical task decomposition using deep reinforcement learning for pick and place subtasks," in *2021 20th International Conference on Advanced Robotics (ICAR)*. IEEE, 2021, pp. 640–645.

[31] Y. Zhen *et al.*, "Robot task planning based on large language model representing knowledge with directed graph structures," 2023. [Online]. Available: https://arxiv.org/abs/2306.05171

[32] N. Shinn *et al.*, "Reflexion: Language agents with verbal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[33] D. Höller *et al.*, "Hddl: An extension to pddl for expressing hierarchical planning problems," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 06, 2020, pp. 9883–9891.

[34] Y. Ding *et al.*, "Robotic task oriented knowledge graph for human-robot collaboration in disassembly," *Procedia CIRP*, vol. 83, pp. 105–110, 2019, 11th CIRP Conference on Industrial Product-Service Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2212827119304263

[35] J. Bai *et al.*, "A dynamic knowledge graph approach to distributed self-driving laboratories," *Nature Communications*, vol. 15, 01 2024.

[36] H. Kasaei and M. Kasaei, "Vital: Visual teleoperation to enhance robot learning through human-in-the-loop corrections," 2024. [Online]. Available: https://arxiv.org/abs/2407.21244

[37] H. Liu, S. Nasiriany, L. Zhang, Z. Bao, and Y. Zhu, "Robot learning on the job: Human-in-the-loop autonomy and learning during deployment," 2023. [Online]. Available: https://arxiv.org/abs/2211.08416

[38] M. Raessa *et al.*, "Human-in-the-loop robotic manipulation planning for collaborative assembly," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 4, pp. 1800–1813, 2020.

[39] Y. Emami, K. Li, L. Almeida, W. Ni, and Z. Han, "Human-in-the-loop machine learning for safe and ethical autonomous vehicles: Principles, challenges, and opportunities," *arXiv preprint arXiv:2408.12548*, 2024.

[40] X. Wu, L. Xiao, Y. Sun, J. Zhang, T. Ma, and L. He, "A survey of human-in-the-loop for machine learning," *Future Generation Computer Systems*, vol. 135, pp. 364–381, 2022.