Combining LLM, Non-monotonic Logical Reasoning, and Human-in-the-loop Feedback in an Assistive AI Agent

Tianyi Fu[1] and Brian Jauw[1] and Mohan Sridharan[1]

***Abstract*— Large Language Models (LLMs) are considered state of the art for many tasks in robotics and AI. At the same time, there is increasing evidence of their critical limitations such as generating arbitrary responses in new situations, inability to support rapid incremental updates based on limited examples, and opacity. Toward addressing these limitations, our architecture leverages the complementary strengths of LLMs and knowledge-based reasoning. Specifically, the architecture enables an AI agent assisting a human to use an LLM to provide generic abstract predictions of upcoming tasks. The agent also reasons with domain-specific knowledge, recent history of interactions with the human, and semantic databases to: (a) provide contextual prompts to the LLM; and (b) compute a plan of concrete actions that jointly implements the current task and prepares for the anticipated task, replanning as needed. Furthermore, the agent solicits and uses high-level human feedback based on need and availability to incrementally revise the domain-specific knowledge and interactions with the LLM. We ground and evaluate our architecture's abilities in the realistic *VirtualHome* simulation environment, demonstrating a substantial performance improvement compared with just using an LLM or an LLM and logical reasoner. Project website: https://brianej.github.io/igfmrdskaa.github.io/**

I. Motivation

Consider an AI agent assisting a human in daily living tasks such as fetching a book or preparing breakfast; Figure 1 shows some snapshots in a simulation environment. Recent generative AI systems such as Large Language Models (LLMs) and Vision Language Models (VLMs) are considered state of the art for the corresponding perception, reasoning, and interaction problems [1]–[5]. At the same time, there is increasing evidence of the limitations of using such "end-to-end" systems for these problems [6], [7]. In particular, they are based on a narrow set of representations and update processes; make arbitrary decisions in truly novel situations; do not support rapid, incremental updates from limited examples; and are opaque [6], [8], [9]. Researchers have tried to address these limitation through frameworks combining knowledge-based and data-driven methods, e.g., using domain knowledge to guide LLMs, and using LLMs to provide high-level tasks implemented by other tools [10]–[12]. However, designing a framework that effectively leverages the strengths of generative models, prior knowledge, and human feedback remains an open problem.

In a departure from existing work, we present an architecture inspired by research in cognitive systems [13] for an AI agent collaborating with a human. As outlined in Figures 1-2, the architecture enables the agent to:

- Automatically identify task-, domain-, and user-specific context for any given task based on semantic knowledge related to the task's expected outcomes, using this context and recent history of the user's commands to prompt an LLM to predict upcoming tasks.
- Leverage prior knowledge and human feedback based on need and availability to adapt the LLM's predictions to the tasks and domain at hand, using non-monotonic logical reasoning to compute and execute a plan of actions that implement current and anticipated task(s).
- Use knowledge of tasks, observation of task outcomes, and human input to provide feedback to the LLM, incrementally revising the LLM's predictions to match the user's preference, leading to faster and more reliable adaptation with fewer interactions.

We implement and evaluate our architecture in the physically realistic VirtualHome [14] simulation environment, using ChatGPT-4o as the LLM [15], Answer Set Programming (ASP) for non-monotonic logical reasoning [16], and WordNet for generic semantic knowledge [17]. Experimental

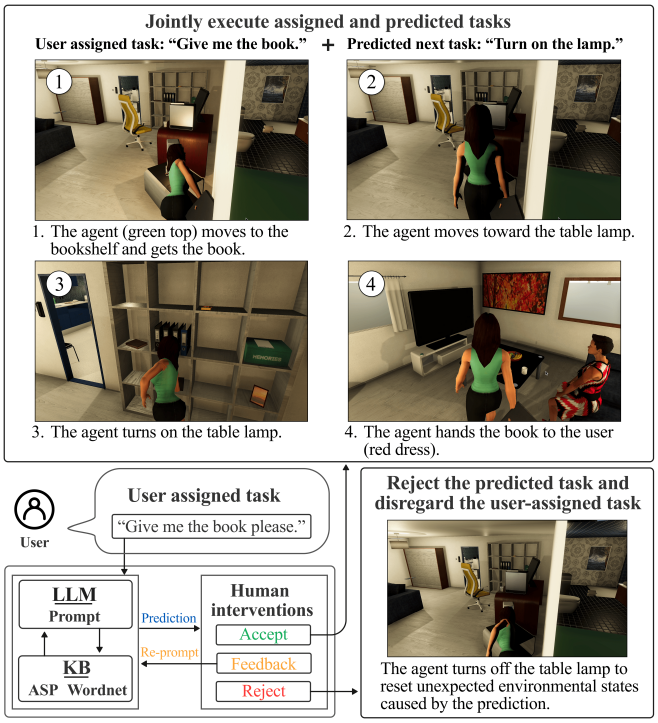

Fig. 1. Embodied AI agent adapts generic, contextual prediction of future task using domain-specific knowledge and human feedback, planning and executing actions to jointly achieve these tasks.

[1]Tianyi Fu, Brian Jauw and Mohan Sridharan are with the Institute of Perception, Action, and Behavior in the School of Informatics at the University of Edinburgh, 10 Crichton Street, Edinburgh EH8 9AB, UK. tianyi.fu@ed.ac.uk, b.e.jauw@sms.ed.ac.uk, m.sridharan@ed.ac.uk

results demonstrate how each component of our architecture and the underlying methodology contribute to significantly improving performance in complex scenarios.

II. Related Work

We motivate our architecture by reviewing related work in data-driven task prediction and planning.

A. High-Level Task Prediction with LLMs

Previous research has shown experimental evidence of an LLM's ability to predict tasks in household environments based on limited instructions or examples [10], [18], [19], although relying on just high-level domain details and static contextual information can produce outdated predictions in practical domains [2], [20], [21]. Although some studies extract contextual information from the environment and use them with LLMs, they do not fully leverage prior domain knowledge [18], [20]. Drawing on these insights, our framework employs semantic similarity analysis to dynamically update domain-specific data used to prompt LLMs, thus offering more accurate predictions.

B. LLMs with External Tools for Task Planning

There is increasing evidence that directly using LLMs as planners can result in poor performance in complex domains [2], [3], [6]. Researchers have instead explored using LLMs to generate Planning Domain Definition Language (PDDL) domain models for general-purpose planners, with humans providing high-level oversight and corrections [7], [11]. Others have used LLMs to generate high-level tasks for achieving a goal, with the tasks being implemented as a sequence of actions by a classical planner with fixed domain knowledge [6], [10]. More recent work that considers domain-specific information and some human feedback to correct LLM predictions does not fully leverage contextual input or support human-in-the-loop (HITL) planning [22]. In contrast, our framework incrementally uses domain knowledge (e.g., plan outcomes) to adapt the prompts to an LLM for predicting high-level tasks. In addition, the LLM's outputs are automatically corrected and incrementally mapped to domain-specific information, leading to more reliable predictions and faster task completion.

C. Human-Centered Feedback

In complex domains, human feedback plays a vital role in ensuring reliable and efficient operation in the context of planning, learning, and other tasks [23]–[25]. Frameworks like TaskBench [26] and Reflexion [27] highlight the importance of iterative human feedback for improving an LLM's performance in complex tasks. Additionally, hierarchical task structures often enhance feasibility of completing multi-step tasks [28]. Our framework employs iterative fine-tuning of LLMs by automatically integrating domain-specific knowledge and HITL feedback [29], [30] based on need and availability; the AI agent can solicit specific feedback if it is not able to make progress despite multiple attempts, and humans can decide to intervene or provide feedback whenever they notice a need to do so.

III. Problem Formulation and Framework

Consider an embodied AI agent assisting a human in a home in performing tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$ that are assigned to the robot one at a time. Each τ_i describes a high-level task (e.g., "heat milk," "prepare a snack") that is implemented as a sequence of fine-grained actions $\{a_1, \ldots, a_{m_i}\}$. Our architecture for this agent combines the complementary strengths of an LLM, knowledge-based reasoning, and HITL feedback, as shown in Figure 2. Specifically, on receiving a user-assigned task, the agent determines contextually relevant information (Section III-B). The information determines the prompt to the LLM to predict the next task(s) (Section III-C). ASP-based non-monotonic logical reasoning is used to jointly compute a plan of actions for the current and predicted tasks. The plan is executed, with replanning to deal with unexpected outcomes (Section III-A). Any input from the user about the predicted task is used by the agent to revise the prediction directly or through the LLM; also, the user can intervene during or after plan execution to set a new task (Section III-D). In addition, the agent can solicit and use human input to address unexpected outcomes from the LLM or after action execution. We describe the components below.

A. Knowledge Representation and Reasoning with ASP

In our architecture, any domain's transition diagram is represented using action language $\mathcal{AL}_d$ [31]. Action languages are formal models of parts of natural language for describing dynamic systems. The domain representation comprises system description $\mathcal{D}$, a collection of statements of $\mathcal{AL}_d$, and a history $\mathcal{H}$. $\mathcal{D}$ has a sorted signature Σ with basic sorts, and domain attributes (statics, fluents) and actions described in terms of the basic sorts.

Our domain includes basic sorts such as $room$, $container$, $furniture$, $food$, $human$, and $step$ (for temporal reasoning) that are arranged hierarchically, e.g., $fruit$ is a sub-sort of $food$ that is a sub-sort of $item$. Statics are domain attributes whose values cannot change, e.g., $next\_to(kitchen, study)$ and fluents are attributes whose values can change. Fluents can be *inertial*, which obey inertia laws and are changed by actions, e.g., $loc(item, room)$ and $in\_hand(agent, item)$; and *defined*, which do not obey inertia laws and are not directly changed by actions, e.g., $otherloc(human, room)$ is the human's location. Actions include agent's actions, e.g., $move(agent, room)$, $pickup(agent, item)$, and $switchon(agent, appliance)$, or exogenous actions, e.g., $exo\_pickup(human, item)$.

Based on Σ, the domain's dynamics are described using three types of axioms such as:

$$move(A, R) \textbf{ causes } loc(A, R) \tag{1a}$$

$$\neg(at(A, R_1), I) \textbf{ if } at(A, R_2), I), R_1 \neq R_2 \tag{1b}$$

$$\textbf{impossible } give(A, O, U) \textbf{ if } loc(A, R_1), \; otherloc(U, R_2), R_1 \neq R_2 \tag{1c}$$

where Statement 1(a), a *causal law*, implies that an agent (A) moves to a room (R), it is located in R; Statement 1(b), a *state constraint*, implies that an agent (A) cannot be in

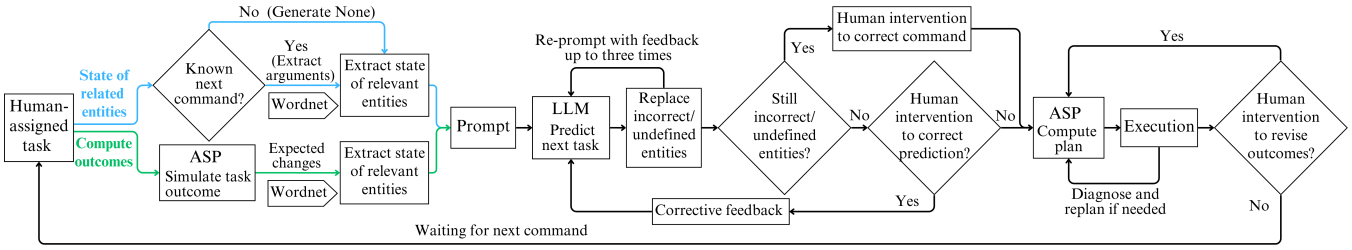

Fig. 2. Architecture overview: (a) any user-assigned task is used to compute contextually-relevant domain-specific information based on command history and expected outcomes, leading to a prompt to the LLM that predicts the subsequent task(s); (b) LLM's output is corrected and adapted to specific domain based on domain knowledge and human feedback; (c) ASP-based reasoning is used to compute and execute a plan of actions to jointly achieve current and future task; (d) Human feedback can revise outcomes and determine next step.

two places ($L1, L2$) at the same time; and Statement 1(c), an *executability condition*, prevents the agent from trying to give an object to a human user who is not in the same room.

History $\mathcal{H}$ is a record of statements of the form $obs(fluent, boolean, step)$, which represent observations, and of the form $hpd(action, step)$, which represent executed actions, at specific time steps.

To reason with knowledge, we automatically construct ASP program $\Pi(\mathcal{D}, \mathcal{H})$ that includes statements from $\mathcal{D}$ and $\mathcal{H}$, inertia axioms, reality check axioms, closed world assumptions for defined fluents and actions, helper relations to reason over time steps, e.g., $holds(fluent, step)$ and $occurs(action, step)$ imply (respectively) that a fluent is true and an action is part of a plan at a time step, and helper axioms to define goals and guide planning and diagnosis. ASP is based on stable model semantics, and encodes *default negation* and *epistemic disjunction*; unlike "$\neg a$" that states *a is believed to be false*, "$not\ a$" only implies *a is not believed to be true*. Each literal is true, false, or unknown, and the agent only believes what it is forced to believe. ASP supports non-monotonic reasoning, the ability to revise previously held conclusions, which is essential for agents operating with incomplete knowledge and noisy observations. All reasoning tasks, i.e., planning, diagnostics, and inference are reduced to computing *answer sets* of Π; we use the SPARC system [32]. For example programs, please see [33].

Since our evaluation scenarios in *VirtualHome* can be complex, with many objects, containers, and locations, computing a plan with multiple steps can often be computationally expensive. To address this issue, we adapt prior work in our group on a refinement-based architecture that represents and reasons about the domain at two different (but formally linked) abstractions [34]. For more complex scenarios, our embodied AI agent can reason, for example, in terms of rooms and regions within rooms. In general, a common criticism of knowledge-based methods is that they need comprehensive domain knowledge, but existing work includes methods that can use incomplete knowledge and revise it over time [35]. Also, the effort involved in encoding prior knowledge is much less than the effort required to train purely data-driven systems.

B. Identifying Domain-specific Context

When a task τ is assigned to the embodied agent by a human user, the agent determines contextually relevant prior information, i.e., a subset of properties of a subset of objects and actions, based on two strategies. First, it searches through recent history H of commands from this user to identify potential next commands C. It processes entries in C and extracts suitable key words K corresponding to actions, objects, and object properties and identifies entities in the domain R semantically similar to entries in K based on the WordNet [17] database; this similarity is computed based on categories (i.e., sorts) of objects and actions. The state of entries in R is considered to be contextually relevant to τ.

The second strategy uses the prior knowledge encoded in the ASP program Π and the associated knowledge base (KB) to simulate the completion of τ and identify set of objects O whose state will change if τ is completed. The agent computes the current state of each entity in O as information contextually relevant to τ and used for further analysis.

C. LLM-based Prediction

In our architecture, ChatGPT-4o is prompted with automatically identified domain-, task-, and user-specific contextual information to predict the subsequent tasks. In a household environment, if the prompts do not consider such contextual information, e.g., current state and user preference to "make coffee" followed by "read newspaper", the LLM will generate inconsistent outputs, e.g., "switch on lamp" when the human has already completed the task, or "make tea" when the human wants coffee. As stated in Section III-B, the contextual information in the prompt includes: likely next task(s) based on user history, current state of relevant entities, and domain-specific knowledge of relevant axioms. The predicted output from LLM is further adapted to the specific domain, using knowledge in ASP program Π and WordNet to replace incorrect actions and objects with appropriate (and semantically similar) entities in the domain, e.g., replace non-existent "mug" with "cup" to serve hot chocolate to human.

D. Human-in-the-loop Revisions

The LLM-generated predictions of subsequent tasks may not always align with user intentions, and the computed (and executed) plan of fine-grained actions may not achieve the user intentions that may have evolved (or changed) over time.

Our architecture uses HITL strategies to address these issues. Specifically, once the LLM predicts the next task, the AI agent communicates the current and predicted task through a visual interface and pauses briefly to see if the human wants to intervene. If no human input is received, the agent proceed to computing and executing the plan for these tasks. If the human does intervene, they can revise or reject the predicted task; in this case, the agent uses the revised input as the goal to be achieved by ASP-based reasoning.

The human can also intervene during or after plan execution. This can take the form of asking the agent to revert to the state before the plan or some steps of the plan were executed, e.g., the agent fetched a newspaper that the human no longer wants to read. If possible, the robot completes this user request or indicates its inability to do so, e.g., the agent cannot undo coffee after it has been prepared. The human input can also take the form of an entirely different task or a revision of their preferences, which the agent considers in the next cycle. Overall, our architecture leads to more natural interactions between the agent and a human.

IV. Experimental Setup and Results

We experimentally evaluated four hypotheses regarding our architecture's capabilities:

H1: Using contextually relevant information to prompt the LLM leads to more accurate future task prediction compared with not using this information.

H2: Leveraging domain knowledge and user feedback significantly reduces effort for improving the LLM's predictions and completing the tasks.

H3: Incrementally adapting LLM predictions to user needs and preferences leads better performance compared with baselines that do not support such adaptation.

H4: Planning actions jointly for the current task and the predicted (i.e., anticipated) task leads to better performance than executing tasks one by one.

We describe the experimental setup and results below.

A. Experimental Setup

We created a simulated household environment in VirtualHome with 36 objects, four rooms, and 11 actions. We created 200 high-level tasks (e.g., rearranging objects in specific configurations, fetching specific objects), each annotated with a textual description and an ASP-based goal, e.g., "has(user, book)" or "on(cup, table)". We considered different scenarios with different initial conditions by varying the state of specific objects, e.g., a lamp switched on or off, and a book on the desk or in the user's hand. As with other such simulated domains, the agent has knowledge of the domain's state (i.e., full observability).

We employed ChatGPT-4o as the LLM for generating task predictions with Chain of Thought (CoT) prompting; as discussed later, prediction accuracy improved when prompts included contextual knowledge and human feedback. To capture realistic task flows, five human participants not involved in our architecture's design were asked to order 20 sequences of 10 tasks under different scenarios, yielding a total of 1000 tasks to be completed by the AI agent. These sequences reflected common household routines and different user preferences, e.g., eating fruit before drinking milk, or drinking tea after selecting a book to read.

For evaluating **H1-H3**, we considered task sequences from one participant at a time to explore the evolution of the user's preferences and behavior. Out of 20 task sequences for any user, some were used as historical knowledge and the rest were used for testing. For any test sequence, the LLM received the first task as input and predicted each subsequent task until the sequence was executed; this evaluation was repeated a certain number of times. Details of number of historical sequences and repetitions in each experiment are provided when summarizing the corresponding results. For hypothesis **H4**, we considered the task sequences of each user, but provided the correct predicted task as input at each step to focus on the planning ability.

B. Baselines

For hypotheses **H1-H3**, we considered three baselines.

(B1) LLM. The LLM independently predicts the next high-level task and generates the corresponding low-level execution plan; non-monotonic logical reasoning (ASP), domain-specific context, or human feedback are not used for validation or correction. Only re-prompting is available as a strategy in response to a predicted task being invalid or undefined.

(B2) LLM, ASP, reprompt. The task predicted by the LLM is implemented as a plan of actions by ASP. There is no correction of LLM output with domain knowledge and no human feedback. Only re-prompting is available if a predicted task is invalid or undefined.

(B3) LLM, ASP, domain-specific correction. B2 is extended by using prior knowledge to automatically adapt the LLM's output to the task and objects defined in the domain before planning an action sequence. If this strategy does not result in a task that can be completed by the agent, the LLMs are re-prompted; any persisting failures are handled through human feedback.

Recall that our architecture extends **B3** by automatically generating contextual prompts (Section III-B) for the LLM; soliciting human feedback (if available) when domain knowledge and re-prompting do not lead to an executable task; using ASP-based diagnosis and replanning when plan execution leads to an unexpected outcome; and soliciting human input to address persistent execution failures. For comparison with baselines, each specific experimental run with our architecture was repeated for each baseline with the same initial condition, historical sequences, and initial task. For **H4**, we used two baselines: **B4**, a variant of **B1** that used an LLM to generate a plan for one task at a time; and **B5**, which used ASP to plan actions for one task at a time.

C. Evaluation Measures

We used five performance measures for evaluation.

1) **Prediction accuracy.** Measures extent to which LLM's predictions align with user's actual input (H1, H2).

2) **Zero-reprompting accuracy.** Measures frequency of LLM's predicted task being correct on the first attempt, i.e., initial prediction accuracy (H1, H2, H3).
3) **Task completion rate.** Measures proportion of predictions successfully implemented and executed as a plan of actions in the simulator (H1, H2).
4) **Average count of re-prompting.** Quantifies iterations required for correct task prediction (H2).
5) **Average number of actions.** Quantifies the number of actions needed to complete the assigned task(s); also computed for task sequence (H4).

D. Experimental Results

In the first experiment, the 20 sequences for any given user were split into four blocks of five sequences each. In each experiment trial, one block was used as historical information and the remaining three blocks (i.e., 15 sequences) were used for testing. We ran four such trials, each with a different block (of five sequences) as historical information, and the entire experiment (with four trials) was repeated three times for the user under consideration. Table I summarizes the results comparing our architecture with **B1-B3**. The use of reprompting in **B2** improved performance, e.g., better task completion rate and slightly better prediction accuracy, and the use of domain-specific knowledge to revise the LLM's predictions in **B3** significantly improved performance, e.g., prediction accuracy and task completion rate, compared with **B1-B2**. Our proposed architecture led to significantly better performance on all measures compared with all baselines, with high prediction accuracy and task completion rate along with significantly reduced need for reprompting. These findings support **H1**, **H2** and **H3**.

To evaluate the impact of historical knowledge, we repeated the experiment mentioned above for different number of historical sequences (0, 5, 10, 15, 19), and computed the prediction accuracy, as summarized in Figure 3. Note that the prediction accuracy only reached a low value for **B1-2**; it reached a much higher value with **B3** but was consistently highest for our architecture.

To further evaluate the impact of contextual knowledge, knowledge-based reasoning, and human feedback (i.e., **H2**), we conducted another experiment. Out of the 20 sequences for any given user, three blocks (each with five sequences) were used as historical context, with the remaining block (of five sequences) used for testing in an experimental trial. We ran four such trials, each with a different block used for testing, and the entire experiment was repeated 10 times for the user. To simulate the presence of noisy historical information, we intentionally replaced 20% of each sequence (i.e., two of 10 tasks) in history with information semantically similar to (in terms of relations or their arguments) but not a correct match to the domain-specific knowledge. For example, *grasp(agent, object)* may replace *pickup(agent, object)*, or *light* may replace *table-lamp*. Table II summarizes the results. The introduction of noise had a negative impact on performance, even with the much larger amount of historical information. However,

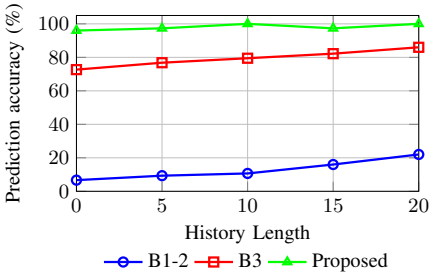

Fig. 3. Prediction accuracy as a function of the number of sequences in historical information. Increasing the number of historical sequences increases accuracy up to a point. The accuracy stabilizes as a low value for **B1-3**; it is reasonably steady at a much higher value for **B3** but it is consistently highest for architecture.

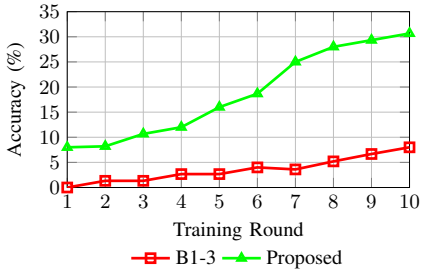

Fig. 4. **Zero-reprompting accuracy** over 10 training rounds of replacing one sequence in the incorrect user's historical information with a correct sequence. Baselines **B1-3** (in red) show minimal adaptation to the changes in historical information, but our architecture quickly leverages the incremental feedback, achieving higher accuracy with minimal need for reprompting.

as before, the use of domain-specific information and the selective use of human feedback improved performance, with our architecture achieving significantly better performance than the baselines and that in Table I with fewer historical sequences but no noise. These results further support **H2-H3**.

Next, we evaluated the architecture's ability to adapt incrementally to changes in user habits (i.e., **H3**) using the sequences of a pair of humans, say user1 and user2, split (as before) into four blocks that each had five sequences. In an experiment trial, we used one block (of five sequences) of user2 for testing. The historical data provided to the LLM had three blocks of sequences (i.e., 15 sequences) from user1, but one historical sequence was replaced by a sequence from user2 (that is not being used for training). The number of such replaced sequences was increased by one over 10 such trials, with the entire experiment repeated three times. We used zero reprompting accuracy as the performance measure, with the results summarized in Figure 4. Baselines **B1-3** (in red) struggle to adapt to the changes in historical sequences, but our architecture quickly leveraged the incremental feedback, achieving higher accuracy with minimal need for reprompting. These results support **H3**.

Finally, we evaluated the benefit of planning jointly for current and anticipated tasks (**H4**). We considered 20 sequences from one (randomly selected) user. For each sequence, **B4** and **B5** produced and executed a sequence of concrete actions for one task at a time using LLM and ASP respectively. With our approach, the agent received the current task and the predicted next task as goals (until end of sequence), and planned and executed a sequence of action to

Architectures →	B1: LLM	B2: LLM, ASP, reprompt	B3: LLM, ASP, corrections	Proposed
Prediction accuracy (in %) ↑	13.78	14.78	76.00	**92.89**
Zero-reprompting accuracy (in %) ↑	5.44	4.67	8.11	**21.56**
Average reprompting count (Num)	1.1129	1.2857	1.0365	**0.8254**
Task completion rate (in %) ↑	7.87	14.44	75.67	**92.78**

TABLE I. Comparing our architecture with **B1-B3**; up and down arrows (↑, ↓) indicate whether higher or lower values are desirable for specific measures. The proposed architecture significantly enhances the prediction accuracy, zero-reprompting accuracy, and task completion rate; statistical tests conducted at 99% level of significance.

Architectures →	B1: LLM	B2: LLM, ASP, reprompt	B3: LLM, ASP, corrections	Proposed
Prediction accuracy (in %) ↑	12.00	12.50	87.60	**94.00**
Zero-reprompting accuracy (in %) ↑	4.80	4.60	8.50	**23.60**
Average reprompting count (Num) ↓	1.1333	1.16	1.0331	**0.7979**
Task completion Rate (in %) ↑	7.2	12.30	87.40	**94.00**

TABLE II. Comparing our architecture with **B1-B3**; up and down arrows (↑, ↓) indicate whether higher or lower values are desirable for specific measures. Using domain-specific knowledge to correct LLM's predictions improves performance (see B3 column). The use of contextual prompts, domain knowledge-based corrections, and selective human feedback in our architecture significantly improves performance in comparison with all baselines; statistical tests conducted at 99% level of significance.

Architectures →	B4: LLM (one task)	B5: ASP (one task)	ASP (joint planning)
Total number of actions for all sequences	779	761	**659**
Average of ratio of number of actions per sequence	1.19	1.16	**1.00**
Average of ratio of number of actions per task	1.32	1.20	**1.00**
Task Completion Rate (%)	64.45%	96.50%	98.50%

TABLE III. Baselines **B4-5** plan actions for one task at a time with LLM and ASP respectively. Our architecture plans actions to jointly achieve current task and prepare for the upcoming tasks, resulting in shorter plans. **B4** could not find suitable action sequences for many sequences; performance measures were only computed for tasks that all three methods completed. Statistical tests at 99% level of significance.

jointly achieve these goals. We only repeated the experiment five times with **B4**, and explored the same across different users. Table III summarizes the results. Our architecture, which planned actions to jointly achieve current task and prepare for the upcoming tasks, resulted in shorter plans. Also, since the number of actions can vary substantially based on the task, the measures computing an average considered the ration of the values for the baselines withe respect to those for our architecture. Note that just using an LLM (**B4**) could not find plans that executed successfully for many sequences, and the performance measures in the top three rows were only computed for tasks that all three methods completed. These results provide support for **H4**. Please see project website [33] for code, results and supporting video.

V. Execution Traces

We present two execution traces to illustrate our architecture's capabilities.

A. Execution Example 1: Context and ASP Planning

Consider the scenario: a robotics book is on the bookshelf in the living room; a table lamp in the living room is turned off; a human is sitting on a chair at the desk in the living room; the agent is also in the living room. When instructed: `"Please give me the book"`, the agent uses ASP-based reasoning to mentally simulates the execution of the following plan to achieve the goal `has(user1, rbook)`: *occurs(move(agent1, bookshelf), 0), occurs(pickup(agent1, rbook, bookshelf), 1), occurs(move(agent1, user1), 2), occurs(give(agent1, rbook, user1), 3).*

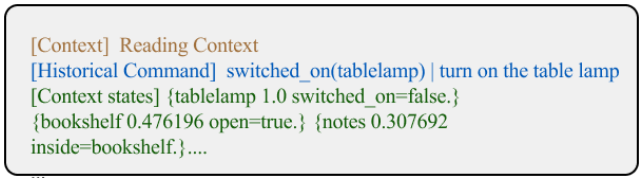

Fig. 5. User input, context, and historical record in the prompt.

The domain objects impacted by this plan are identified: *r-book*, *bookshelf*. Next, using the semantically relevant data from WordNet, the agent identifies the context most relevant to this task as *ReadingContext*; other contexts related to dining and cleaning are considered less relevant. The agent also examines recent history of tasks (of this user) co-occurring with the reading context and identifies the related command *please turn on the table-lamp* and the related object as *table-lamp*. It provides all of these as related context to the LLM, as shown in Fig 5, which responds with:

> *Predicted command*: `switchon tablelamp`
> *Reason*: The suggested command from history is to switch on the table lamp, which aligns with the user's frequently executed actions and supports a reading context by providing adequate lighting.

This is translated to a goal for ASP-based reasoning: `holds(switched_on(tablelamp), I)`. The agent then gives the user a brief period to review and revise the goal. Since there is no human interrupt, the agent uses ASP-based reasoning to compute a plan of actions for both the goals, which is then executed in the simulator—see Fig 6, with the corresponding outcomes revising domain state.

```
Current command: has(user1, rbook)
Predicted command from LLM: switchon tablelamp
ASP goals: goal_2(I) :- holds(switched_on(tablelamp), I).

Solution found with n=7

----- OCCURS OUTPUT -----
{occurs(move(agent1,livingroom),0),
occurs(move(agent1,bookshelf),1),
occurs(pickup(agent1,rbook,bookshelf),2),
occurs(move(agent1,user1),3),
occurs(give(agent1,rbook,user1),4),
occurs(move(agent1,tablelamp),5),
occurs(switchon(agent1,tablelamp),6)}

Convert to the action script for simulator...
<char1> [move] <bookshelf> (bookshelf_id)
<char1> [pickup] <rbook> (rbook_id) <bookshelf> (bookshelf_id)
<char1> [move] <user1> (user1_id)
<char1> [put] <rbook> (rbook_id) <user> (user_id)
<char1> [move] <tablelamp> (tablelamp_id)
<char1> [switchon] <tablelamp> (tablelamp_id)
```

Fig. 6. ASP generated plan for goals in Execution Example 1.

B. Execution Example 2: Prediction Error, HITL, Diagnosis

Consider another scenario: the human and the agent are in the living room; a coffee table is located in the adjacent bedroom; food items such as cereal, cutlets and juice are laid out on the kitchen counter. The human is interacting with the agent as part of a lunchtime routine. Recall that our architecture supports two strategies to handle situations in which the LLM's prediction do not match domain knowledge or user preferences (Sections III-B-III-D).

Contextual replacement of LLM output. If LLM prediction includes an entity not in the ASP-based description of domain knowledge, the agent uses semantic knowledge in WordNet and prior context to identify the closest match. For example, when LLM's prediction requires the agent to fetch *cornflakes*, which is not defined in the domain, the agent identifies *cereal* from domain knowledge as a close match.

User feedback. When the LLM's prediction does not match the user's preference, the user can ask for it to be revised (Figure 7), replaced, or ignored, supporting human oversight.

ASP-based Diagnosis. During plan execution, the agent is unable to pickup *coffee* from the *coffeetable*—see Figure 8. This triggers ASP-based diagnosis to identify that coffee is no longer available in the domain (e.g., it has been consumed). The agent reasons and offers an alternative plan to fetch *juice* (a different beverage) instead. The corresponding plan results in successful execution, with the domain state being updated accordingly.

VI. Conclusion

This paper presented an architecture that integrates generic knowledge (LLM), reasoning with domain-specific knowledge (ASP), contextual information, and human-in-the-loop (HITL) feedback to enable an embodied AI agent to assist a human in a complex domain. Experimental evaluation in

```
Attempt 1:
Predicted command from LLM: give cutlets to user.
Reason: Recent history of user command suggests the user will want
cutlets.
ASP Goal: goal(I) :- holds(has(user1, cutlets), I).

A suitable ASP goal was found. Accept this goal? (yes/no): no
Using user feedback => I want something to drink.

Attempt 2:
Predicted command from LLM: give juice to user.
Reason: User preference is to have something to drink and juice is
available in the domain.

ASP Goal: goal(I) :- holds(has(user1, juice), I).
```

Fig. 7. Feedback adjusts LLM prediction in Execution Example 2.

```
Step 1 =>  <char1> [move] <coffeetable> (coffeetable_id)
[Step 1] => success
Step 2 =>  <char1> [pickup] <coffee> (coffee_id) <coffeetable>
(coffeetable_id)
[Step 2] => failed
[DIAG] Start diagnosing => ScriptExcutor 1:  Script is impossible to
execute
[DIAG] diagnosis.sp: #item => ['coffee'], wrote 2 facts.
Result:{expl(removedItem,1)}.
[DIAG] top_context=DiningContext, top_items=[ {name: juice,
weight: 0.43}...]
[DIAG] Replaced coffee with juice.
[DIAG] re-sim with replaced item =>
Step 1 =>  <char1> [move] <kitchencounter> (kitchencounter_id)
[Step 1] => success
Step 2 =>  <char1> [pickup] <juice> (juice_id) <kitchencounter>
(kitchencounter_id)
[Step 2] => success
[Step 2] Knowledge REMOVED => (juice ON, kitchencounter)
[Step 2] Knowledge ADDED   => (character, HOLDS_RH, juice)
....
Updating domian knowledge from solution results...
REMOVED = {on(coffeetable, coffee)}
```

Fig. 8. ASP-based diagnosis and correction of execution errors.

the VirtualHome environment indicates that computing and using contextual information leads to more reliable future task predictions from the LLM, and reduces the need for re-prompting. Also, ASP-based reasoning with domain knowledge to plan and execute a sequence of actions that jointly achieve current task and prepare for anticipates task leads to performance improvement. In addition, HITL feedback enables real-time error correction and incremental knowledge refinement. Overall, our architecture improves performance substantially compared state of the art data-driven and/or knowledge-based baselines.

Our architecture opens up multiple directions for further research. This includes exploring the architecture's applicability to more complex tasks and domains. We will also investigate the expansion of the architecture to consider reasoning with (learned) probabilistic models of uncertainty by building on prior work on a refinement-based architecture [34]. In addition, we will consider expanding the

architecture's learning capabilities to incrementally acquire and revise domain knowledge [36]. Furthermore, we will implement and evaluate the architecture's capabilities on physical robot platforms. The long-term goal is to develop intelligent agents capable of seamlessly reasoning with prior knowledge, learning from experience, interacting with humans, and adapting their behavior on the fly to evolving human preferences and environments.

REFERENCES

[1] R. Doshi, H. Walke, O. Mees, S. Dasai, and S. Levine, "Scaling Cross-Embodied Learning: One Policy for Manipulation, Navigation, Locomotion, and Aviation," in *International Conference on Robot Learning*, 2024.

[2] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *International conference on machine learning*, 2022.

[3] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone, "LLLM+P: Empowering Large Language Models with Optimal Planning Proficiency," *arXiv preprint arXiv:2304.11477*, 2023.

[4] H. Zhang, W. Du, J. Shan, Q. Zhou, Y. Du, J. B. Tenenbaum, T. Shu, and C. Gan, "Building Cooperative Embodied Agents Modularly with Large Language Models," in *International Conference on Learning Representations*, May 7-11, 2024.

[5] Z. Zhao, W. S. Lee, and D. Hsu, "Large Language Models as Commonsense Knowledge for Large-Scale Task Planning," in *International Conference on Neural Information Processing Systems*, 2023.

[6] S. Kambhampati, K. Valmeekam, L. Guan, M. Verma, K. Stechly, S. Bhambri, L. Saldyt, and A. Murthy, "Llms can't plan, but can help planning in llm-modulo frameworks," *arXiv preprint arXiv:2402.01817*, 2024.

[7] T. Silver, V. Hariprasad, R. S. Shuttleworth, N. Kumar, T. Lozano-Pérez, and L. P. Kaelbling, "Pddl planning with pretrained large language models," in *NeurIPS Foundation Models for Decision Making Workshop*, 2022.

[8] S. Dohare, J. F. Hernandez-Garcia, Q. Lan, P. Rahman, A. R. Mahmood, and R. S. Sutton, "Loss of Plasticity in Deep Continual Learning," *Nature*, vol. 632, no. 8026, pp. 768–774, 2024.

[9] S. Lu, I. Bigoulaeva, R. Sachdeva, H. T. Madabushi, and I. Gurevych, "Are Emergent Abilities in Large Language Models just In-Context Learning?" in *Annual Meeting of the Association for Computational Linguistics*, Bangkok, Thailand, August 11-16, 2024, pp. 5098–5139.

[10] R. Arora, S. Singh, K. Swaminathan, A. Datta, S. Banerjee, B. Bhowmick, K. M. Jatavallabhula, M. Sridharan, and M. Krishna, "Anticipate & act: Integrating llms and classical planning for efficient task execution in household environments," in *International Conference on Robotics and Automation*, 2024.

[11] L. Guan, K. Valmeekam, S. Sreedharan, and S. Kambhampati, "Leveraging pre-trained large language models to construct and utilize world models for model-based task planning," *Advances in Neural Information Processing Systems*, vol. 36, 2023.

[12] T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, "Toolformer: Language Models Can Teach Themselves to Use Tools," in *Advances in Neural Information Processing Systems*, 2023.

[13] M. Sridharan, "Back to the Future of Integrated Robot Systems," in *AAAI Conference on Artificial Intelligence*, 2025.

[14] X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba, "Virtualhome: Simulating household activities via programs," in *International Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8494–8502.

[15] OpenAI, "GPT-4o System Card," *ArXiv*, vol. abs/2410.21276, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:273662196

[16] M. Gelfond and Y. Kahl, *Knowledge Representation, Reasoning and the Design of Intelligent Agents*. Cambridge University Press, 2014.

[17] G. A. Miller, "WordNet: A Lexical Database for English," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[18] B. Chen, F. Xia, B. Ichter, K. Rao, K. Gopalakrishnan, M. S. Ryoo, A. Stone, and D. Kappler, "Open-vocabulary queryable scene representations for real world planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[19] S. Singh, K. Swaminathan, R. Arora, R. Singh, A. Datta, D. Das, S. Banerjee, M. Sridharan, and M. Krishna, "Anticipate & collab: Data-driven task anticipation and knowledge-driven planning for human-robot collaboration," *arXiv preprint arXiv:2404.03587*, 2024.

[20] D. Driess, F. Xia, M. S. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu *et al.*, "Palm-e: An embodied multimodal language model," *arXiv preprint arXiv:2303.03378*, 2023.

[21] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, and H. Wang, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2024.

[22] S. Singh, K. Swaminathan, N. Dash, R. Singh, S. Banerjee, M. Sridharan, and M. Krishna, "AdaptBot: Combining LLM with Knowledge Graphs and Human Input for Generic-to-Specific Task Decomposition and Knowledge Refinement," in *International Conference on Robotics and Automation (ICRA)*, May 2025.

[23] 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 *International Conference on Computer Vision*, 2023, pp. 2998–3009.

[24] S. Amershi, M. Cakmak, W. B. Knox, and T. Kulesza, "Power to the people: The role of humans in interactive machine learning," *AI magazine*, vol. 35, no. 4, pp. 105–120, 2014.

[25] H. Shaik and A. Doboli, "An overview and discussion on using large language models for implementation generation of solutions to open-ended problems," *arXiv preprint arXiv:2501.00562*, 2024.

[26] Y. Shen, K. Song, X. Tan, W. Zhang, K. Ren, S. Yuan, W. Lu, D. Li, and Y. Zhuang, "Taskbench: Benchmarking large language models for task automation," *arXiv preprint arXiv:2311.18760*, 2023.

[27] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[28] D. Höller, G. Behnke, P. Bercher, S. Biundo, H. Fiorino, D. Pellier, and R. Alford, "Hddl: An extension to pddl for expressing hierarchical planning problems," in *AAAI conference on artificial intelligence*, vol. 60, 2020, pp. 9883–9891.

[29] 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," *The International Journal of Robotics Research*, p. 02783649241273901, 2022.

[30] H. Kasaei and M. Kasaei, "Vital: Visual teleoperation to enhance robot learning through human-in-the-loop corrections," *arXiv preprint arXiv:2407.21244*, 2024.

[31] M. Gelfond and D. Inclezan, "Some properties of system descriptions of ald," *Journal of Applied Non-Classical Logics*, vol. 23, no. 1–2, p. 105–120, 2013.

[32] E. Balai, M. Gelfond, and Y. Zhang, "Towards Answer Set Programming with Sorts," in *International Conference on Logic Programming and Nonmonotonic Reasoning*, September 2013.

[33] T. Fu, B. Juaw, and M. Sridharan, "https://brianej.github.io/igfmrdskaa.github.io/," 2025.

[34] M. Sridharan, M. Gelfond, S. Zhang, and J. Wyatt, "REBA: A Refinement-Based Architecture for Knowledge Representation and Reasoning in Robotics," *Journal of Artificial Intelligence Research*, vol. 65, pp. 87–180, May 2019.

[35] H. Dodampegama and M. Sridharan, "Knowledge-based Reasoning and Learning under Partial Observability in Ad Hoc Teamwork," *Theory and Practice of Logic Programming*, vol. 23, no. 4, pp. 696–714, 2023.

[36] M. Sridharan and T. Mota, "Towards Combining Commonsense Reasoning and Knowledge Acquisition to Guide Deep Learning," *Autonomous Agents and Multi-Agent Systems*, vol. 37, no. 4, 2023.