Thinking Together, Apart: Reasoning and Learning in Ad Hoc Teamwork

Hasra Dodampegama HASRA.DODAMPEGAMA@ED.AC.UK
Mohan Sridharan M.SRIDHARAN@ED.AC.UK
Institute of Perception, Action, and Behavior, School of Informatics, University of Edinburgh, UK

Abstract

AI agents deployed to assist and collaborate with other agents often have to do so without prior coordination. Methods considered state of the art for such *ad hoc teamwork* pose it as a learning problem, using a large labeled dataset to model the action choices of other agents (or agent types) and determine the actions of the ad hoc agent. These methods lack transparency and make it difficult to rapidly revise existing knowledge in response to changes. Our architecture for ad hoc teamwork leverages the complementary strengths of knowledge-based and data-driven methods for reasoning and learning. For any given goal, we enables an ad hoc agent to determine its actions through non-monotonic logical reasoning with: (a) prior domain-specific commonsense knowledge; (b) models learned and revised rapidly to predict the behavior of other agents; and (c) anticipated abstract future goals based on generic knowledge of similar situations in an existing foundation model. The agent also processes natural language descriptions and observations of other agents' behavior, incrementally acquiring and revising knowledge in the form of objects, actions, and axioms that govern domain dynamics. We experimentally evaluate the capabilities of our architecture in *VirtualHome*, a realistic simulation environment.

1. Introduction

Consider an assistive AI agent deployed to complete household tasks in collaboration with a human it has not worked with before. Figure 1 shows snapshots of a motivating scenario in which the AI agent and a human agent are preparing breakfast and setting up a workstation. The agents have a limited view of the environment and do not communicate with each other. Each agent is aware of the state of the domain, including the location of teammates and the outcomes of their actions (e.g., change in location of an object moved by a teammate). The agents have to reason with different descriptions of domain knowledge and uncertainty that include qualitative statements (“eggs are usually in the fridge") and quantitative measures of uncertainty (“I am 90% sure I saw the eggs on the kitchen table"), adapting their actions to changes in the domain and teammates' behavior. These characteristics correspond to *Ad hoc Teamwork* (AHT), requiring cooperation “on the fly" without prior coordination (Stone et al., 2010). It arises in many practical applications in robotics, and poses challenges in knowledge representation, reasoning, and learning.

Research in AHT has evolved from using predefined protocols that encoded specific actions for the ad hoc agent in specific situations, to methods that learn probabilistic or deep network models to estimate the behavior of other agents (or agent “types") and optimize the ad hoc agent's actions

Figure 1: Screenshots from *VirtualHome* (Puig et al., 2018) showing a human (female in green top) and an assistive AI agent (male in blue shirt) collaborating.

based on a long history of prior interactions with similar agents. It is often difficult to gather such training datasets of different situations and computationally expensive to build the necessary models. Moreover, these models lack transparency, making it difficult to understand the agent's decisions.

In a departure from existing work, our prior work developed an AHT architecture that enabled an ad hoc agent to make decisions based on non-monotonic logical reasoning with prior knowledge and simple predictive models of other agents' behavior (Dodampegama & Sridharan, 2023). This paper describes an architecture (**KAT**) that extends our prior work to enable each ad hoc agent to:

- Perform non-monotonic logical reasoning with prior domain knowledge and predictive models of other agents' behavior learned from limited examples to support scalable adaptation;
- Leverage a Large Language Model (LLM) to anticipate future tasks to be completed, adapting the LLM's output to domain-specific knowledge and experience; and
- Incrementally revise prior knowledge based on LLM-based processing of natural language descriptions of actions and outcomes, and decision tree induction applied to observations.

We use Answer Set Programming (ASP) (Gelfond & Kahl, 2014) for non-monotonic logical reasoning, GPT4o mini (OpenAI et al., 2024) as the LLM. We explored decision making at different abstractions in household scenarios in *VirtualHome*, a realistic physics-based 3D simulation environment for multiagent collaboration (Puig et al., 2018).

2. Related Work

Researchers have explored AHT for more than two decades under different names (Mirsky et al., 2022). Initial work used predefined protocols or plays that encoded specific actions for the ad hoc agent in specific scenarios (Bowling & McCracken, 2005). Subsequent work used probabilistic and sampling-based methods such as Upper Confidence bounds for Trees to determine the ad hoc agent's actions (Barrett et al., 2013). Recent methods considered state of the art for AHT have posed it primarily as a learning problem; a key component predicts behavior of other agents and determines the ad hoc agent's behavior using a long history of prior interactions with similar agents or agent types. This includes the use of Fitted Q Iteration to learn action selection policies from offline data of each teammate type (Barrett et al., 2017); attention-based recurrent neural networks for real-time adaptation (Chen et al., 2020); graph neural networks to simulate interactions with other agents (agent types) and determine the ad hoc agent's behavior (Rahman et al., 2021); self-play to learn a cooperation policy and candidate teammate policies (Fang et al., 2024); sequential and hierarchical variational auto encoders to model teammates' changing behaviors (Zintgraf et al., 2021); and model-based RL methods to learn separate models of the environment and teammates (Xu et al., 2025). These methods are resource-hungry, requiring substantial computation and training examples to learn models that are often opaque, limiting interpretability of the system.

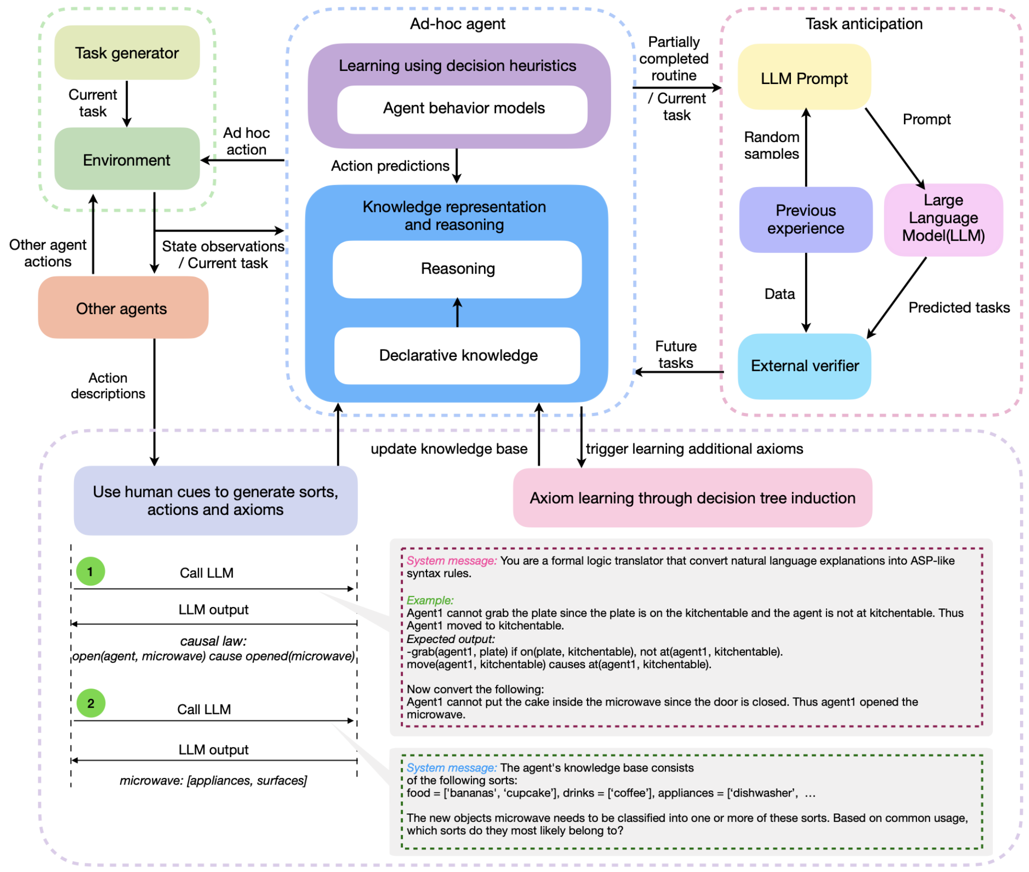
Frameworks based on "foundation" models are considered state of the art for various problems in robotics and AI, and an LLM-based framework has been developed to compute the ad hoc agent's actions (Liu et al., 2024). At the same time, it is known that such models can make arbitrary decisions in novel situations, do not plan, and are more effective when used to generate abstract guidance that is validated before use (Kambhampati et al., 2024).

This paper builds on our proof-of-concept work (Dodampegama & Sridharan, 2023) that enabled an ad hoc agent to reason with domain knowledge and predictive models of other agents in simple domains. Here we consider a more complex household domain, enabling an ad hoc agent to leverage an LLM to anticipate high-level future tasks, use logics to jointly plan for current and future tasks, and use decision-tree induction and an LLM for acquiring previously unknown knowledge from observations and natural language descriptions.

3. Methodology

Figure 2 outlines KAT (***K**nowledge-based Reasoning and Learning for **A**d Hoc **T**eamwork*), our architecture for an ad hoc agent collaborating with other agents (human, AI). An external *task generator* is used to generate a realistic, evolving routine of tasks for any given day (e.g., "make breakfast", "set up workstation", "prepare lunch"), dispatching tasks one at a time to all agents. Each agent is unaware of the task generation strategy and starts with no prior knowledge of the preferences, capabilities, and strategies of other agents, although it expects teammates to collaborate to complete assigned tasks. Each agent receives information about the current state, and computes and executes actions to complete task(s). The ad hoc agent determines its action by reasoning with prior knowledge (Section 3.1) and the actions of other agents predicted by models learned from runtime observations (Section 3.2). It also prompts an *LLM* with recent observations and completed tasks to anticipate future tasks (Section 3.3). It validates and adapts the LLM's output based on domain-specific knowledge, and jointly plans actions to achieve the current and anticipated task(s). In addition, a human (agent) occasionally describes an agent's actions (e.g.,"Agent 1 cannot put the cake in the microwave since its door is closed"). The ad hoc agent uses these descriptions to learn previously unknown domain knowledge in the form of objects, actions, and axioms. Furthermore, it uses observations obtained during plan execution to learn missing axioms based on decision tree induction (Section 3.4). We use the example scenario given below to describe KAT's components for one ad hoc agent and a human; we consider additional agents during evaluation.

Example 1 ***[Human-AI agent collaboration scenario]***
Consider an AI agent and a human agent collaborating to complete daily household tasks such as preparing breakfast or setting up the home work-station (see Figure 1). The agents can interact with the environment through actions that involve moving to places, picking up or placing objects,


Task generator
Current task
Environment
Ad hoc action
Other agent actions
State observations / Current task
Other agents
Action descriptions
Ad-hoc agent
Learning using decision heuristics
Agent behavior models
Action predictions
Knowledge representation and reasoning
Reasoning
Declarative knowledge
Partially completed routine / Current task
Future tasks
Task anticipation
LLM Prompt
Random samples
Prompt
Previous experience
Large Language Model(LLM)
Data
Predicted tasks
External verifier
update knowledge base
trigger learning additional axioms
Use human cues to generate sorts, actions and axioms
Axiom learning through decision tree induction
1
Call LLM
LLM output
causal law:
open(agent, microwave) cause opened(microwave)
2
Call LLM
LLM output
microwave: [appliances, surfaces]

System message: You are a formal logic translator that convert natural language explanations into ASP-like syntax rules.

Example:
Agent1 cannot grab the plate since the plate is on the kitchentable and the agent is not at kitchentable. Thus Agent1 moved to kitchentable.
Expected output:
-grab(agent1, plate) if on(plate, kitchentable), not at(agent1, kitchentable).
move(agent1, kitchentable) causes at(agent1, kitchentable).

Now convert the following:
Agent1 cannot put the cake inside the microwave since the door is closed. Thus agent1 opened the microwave.

System message: The agent's knowledge base consists of the following sorts:
food = ['bananas', 'cupcake'], drinks = ['coffee'], appliances = ['dishwasher', …

The new objects microwave needs to be classified into one or more of these sorts. Based on common usage, which sorts do they most likely belong to?

Figure 2: KAT embeds the principles of refinement and ecological rationality to leverage the complementary strengths of knowledge-based and data-driven reasoning and learning.

switching appliances on or off, and opening or closing appliances. Completing a task requires a sequence of such actions to be computed and executed by members of the team without any direct communication between them. The ad hoc agent assumes that any teammate will have access to the same information about domain state, predicts the actions the teammate will execute over the next few steps, and computes its plan to complete the current task and prepare for the upcoming task(s). Each ad hoc agent's prior commonsense knowledge includes relational descriptions of some attributes of the domain, objects, and human. It also includes axioms governing actions and changes, e.g., each agent is aware that it cannot hold more than two objects at a time.

3.1 Knowledge Representation and Reasoning

In KAT, any given domain's transition diagram is described using an extension of *action language* $\mathcal{AL}_d$ (Gelfond & Inclezan, 2013). Action languages are formal models of parts of natural language for describing transition diagrams of dynamic systems. The domain representation comprises a 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 these sorts. Basic sorts in our example scenario include $object$, $appliance$, ad_hoc_agent, $human$, and $step$ (for temporal reasoning), and are arranged hierarchically, e.g., $apple$ is a sub-sort of $food$, a sub-sort of $graspable$, a sub-sort of $object$. Actions can be $agent\_actions$ such as $grab(ad\_hoc\_agent, object)$, $move(ad\_hoc\_agent, location1, location2)$ that are performed by the ad hoc agent; or $exo\_actions$ (exogenous actions) such as exo_grab $(other\_agent, object)$, $exo\_switch\_on(other\_agent, appliance)$ which are performed by other agents, e.g., human or another AI agent. Statics (fluents) are domain attributes whose values cannot change. Fluents can be *inertial*, which obey inertia laws and are changed by the ad hoc agent's actions, e.g., $at(ad\_hoc\_agent, location)$ is the ad hoc agent's location; or *defined*, which do not obey inertia laws and are not directly changed the by ad hoc agent's actions, e.g., $agent\_at(other\_agent, location)$ is a teammate's location computed by (say) external sensors. Given a specific Σ, the domain's dynamics are described using axioms such as:

$$open(A, E) \textbf{ causes } opened(E) \tag{1a}$$

$$\neg at(A, L1) \textbf{ if } at(A, L2), L1 \neq L2 \tag{1b}$$

$$\textbf{impossible } grab(A, O) \textbf{ if } on(O, E),\ \neg opened(E) \tag{1c}$$

where Statement 1(a), a *causal law*, implies that an agent executing the $open(A, E)$ action causes an appliance E to be opened; Statement 1(b), a *state constraint*, implies that an agent (A) cannot be in two places ($L1, L2$) at the same time; and Statement 1(c), an *executability condition*, prevents the ad hoc agent (A) from trying to grab an object (O) from an appliance (E) that is not open.

The history, $\mathcal{H}$, is a record of statements of the form $obs(fluent, boolean, step)$, which represent observations, and $hpd(action, step)$, which represent executed actions, at specific steps. $\mathcal{H}$ also includes default statements that are true in the initial state.

To reason with knowledge, a script automatically constructs program $\Pi(\mathcal{D}, \mathcal{H})$ in CR-Prolog, an extension to ASP that supports consistency restoring (CR) rules. $\Pi(\mathcal{D}, \mathcal{H})$ contains statements from $\mathcal{D}$ and $\mathcal{H}$, inertia axioms, reality check axioms, closed world assumptions for defined fluents and actions, helper relations such as $holds(fluent, step)$ and $occurs(action, step)$ that imply (respectively) that a fluent is true and an action is part of a plan at a particular time step, and helper axioms that define goals and guide planning and diagnosis. ASP is based on stable model semantics, and encodes *default negation* and *epistemic disjunction*, i.e., unlike "$\neg a$" that states *a is believed to be false*, "$not\ a$" only implies *a is not believed to be true*, and unlike "$p\ \vee\ \neg p$", "$p\ or\ \neg p$" is not tautologous. Each literal is true, false, or unknown, and the agent only believes that which it is forced to believe. ASP supports non-monotonic logical reasoning, i.e., the ability to revise previous conclusions, which is essential for agents operating in practical domains with incomplete knowledge and noisy observations. The CR rules allow the agent to make assumptions under exceptional circumstances to recover from inconsistencies. All reasoning tasks, i.e., planning, diagnostics, and inference are reduced to computing *answer sets* of Π subject to some criteria (e.g., minimize costs) and extracting the action sequence; we do so using the SPARC system (Balai et al., 2013).

Our example scenario is complex, with many objects, containers, and locations, e.g., there can be $\approx 10^{25}$ states with just one ad hoc agent and one human, making it computationally expensive to compute plans with multiple steps. To support scalability, we build on prior work in our group on a refinement-based architecture (Sridharan et al., 2019) to enable the ad hoc agent to represent

Table 1: Attributes used to create the behavior models of the other agents in VirtualHome.

Description of the attribute
Immediate two previous actions of the agent
Position of the agent (x,y,z)
Orientation of the agent (x,y,z)
Objects associated with the goal
Any objects in the hand of the agent
Any objects in the hand of the remaining agents
Current and previous tasks
Flags (weekday, going to office, guests expected)

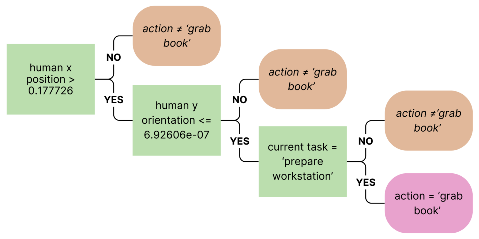


Figure 3: FF tree model of human behavior for the grab_book action in the example scenario from the *VirtualHome* domain.

and reason at two resolutions. Specifically, a fine-resolution description is defined as a refinement of a coarse-resolution description, with the agent now able to reason about aspects of the domain that were previously abstracted away. A common criticism of reasoning methods is that they need comprehensive domain knowledge, but architectures that embed principles such as refinement have demonstrated the ability to reason with the available knowledge and revise it incrementally over time. Also, most of the steps for encoding the knowledge can be automated, and the effort involved in encoding prior knowledge is much less than that needed to train purely data-driven systems.

3.2 Agent Behavior Models

Since reasoning with prior domain knowledge that is incomplete or inconsistent will lead to poor performance, KAT enables the ad hoc agent to reason with models that predict the action choices of other agents that are learned (and revised) rapidly. This capability is achieved by embedding the Ecological Rationality (ER) principle (Gigerenzer, 2020), which is based on Herb Simon's original definition of Bounded Rationality (Simon, 1956) and the algorithmic theory of decision heuristics (Gigerenzer, 2016). ER explores decision making "in the wild", i.e., under open world uncertainty with the space of possibilities not fully known, and characterizes behavior as a joint function of internal cognitive processes and the environment. In the absence of comprehensive knowledge, optimal decisions may be unknowable and not just hard to compute, so ER advocates the use of *adaptive satisficing* and decision heuristics (e.g., tallying, sequencing, fast and frugal methods) to ignore part of the information and make decisions more quickly, frugally, and accurately than sophisticated methods with many more free parameters. This approach has been shown to provide better performance than more sophisticated methods in practical applications (Gigerenzer, 2016).

KAT enables the ad hoc agent to select relevant attributes and rapidly learn (and revise) an ensemble of *Fast and Frugal* (FF) trees from limited data to predict the behavior of each teammate (or teammate type). Each FF tree provides a binary choice for a particular action, and the number of leaves in a tree is limited by the number of attributes (Katsikopoulos et al., 2021). Each level of the tree contains an exit, allowing the agent to make decisions quickly. Also, these models enable the ad hoc agents to consider the more informative attributes and stop as soon as a rational option is found. Figure 3 shows an FF tree learned for a human, and Table 1 shows the attributes used. The initial version of these trees were constructed from only 1000 traces of other agents' actions (guided by simple hand-crafted policies) and domain states. Consistent agreement (or disagreement) between observed outcomes and the predictions provided by these behavior models triggers the use of a particular model for subsequent steps, or leads to the revision of existing model(s), allowing the ad hoc agent to quickly adapt to changes in the domain or an agent's behavior.

3.3 Task Anticipation

As stated earlier, there is increasing evidence that LLMs make arbitrary decisions in novel situations. They are more effective when used as translators between natural and domain-specific languages, and to generate high-level (generic) guidance that is validated externally before being implemented by planning subroutines (Kambhampati et al., 2024). Building on these findings, KAT enables an ad hoc agent to use an LLM to anticipate the high-level future task (e.g., *prepare dinner*) likely to be assigned once the current task is done. Recall that in the absence of the LLM, the agents are informed about the target tasks one at a time. We experimentally demonstrate in Section 4.2 that jointly planning to complete the current task and anticipated task, e.g., fetch some ingredients from the fridge for making lunch while fetching eggs for cooking breakfast, improves team performance. The ad hoc agent uses a combination of three prompting strategies to interact with the LLM:

1. **Adopting persona:** A specific role or character is assigned to guide the LLM's responses to be more (contextually) consistent with the assigned role.
2. **Few-shot prompting:** The prompt includes a few examples of the expected output in specific situations, guiding the use of pretrained knowledge.
3. **Chain-of-thought (CoT):** The prompt includes a step-by-step reasoning process that can be followed to arrive at an answer, leading to more accurate responses.

A 'system message' guides the LLM to adopt the persona of a household assistant and to complete the partially completed routine by selecting potential future tasks from the available list of tasks in the example scenario within the *VirtualHome* domain. With the 'few-shot' prompting approach, the prompt includes two task routines randomly chosen from previous days. Next, CoT prompting is used to explain the reasoning behind each task in the few shot examples. Such explanations can be provided manually by the system designer or generated by the LLM. The system message, few-shot examples with CoT explanation, and the current query (i.e., partially completed or empty task routine for the day) are provided as input to the LLM.

The LLM's output to a prompt is parsed by an *external validator* to check whether the tasks are feasible and in a reasonable order. Specifically, the validator compares the LLM's output with domain- and task-specific contextual features extracted from existing knowledge and recent observations to eliminate tasks that are invalid or irrelevant, and reorder tasks according to the human preferences. For example, the ad hoc agent will prioritize preparation of the workstation over packing a lunch box when the human is working from home. Since the list of validated tasks can change over time, KAT enables the ad hoc agent to consider one anticipated task and the current task as the joint goal for which a plan is to be computed.

3.4 Knowledge Acquisition

Since making decisions based on incomplete or inconsistent knowledge can lead to ineffective collaboration, KAT enables the ad hoc agent to incrementally acquire domain knowledge, reduce ambiguity, and support reliable decision making. The agent acquires knowledge in the form of objects, actions, and axioms using two strategies: (i) it learns from cues provided by a human during task execution; and (ii) it explores actions in different states, identifying and correcting inconsistencies.

Learning from human cues. When an ad hoc agent receives a cue from the human, e.g., "Agent 1 cannot put the cake inside the microwave since the microwave's door is closed", it automatically prompts the LLM to output candidate axioms by combining a predefined system message with two examples of input and expected output, the structure of the three types of axioms (e.g., "a causes f if p"), and the query based on the cue (message 1 in Figure 2). The LLM's output is parsed using regular expressions to discard any output that does not match the expected format of axioms. From the LLM's output that passes this check, the agent extracts actions (verbs), objects, and action preconditions and effects. If there are any new objects that are not already defined in the current ASP program ($\Pi(\mathcal{D}, \mathcal{H})$), the LLM is asked to assign a sort label to these objects by prompting it (message 2 in Figure 2) based on a predefined message template, existing sorts in $\Pi(\mathcal{D}, \mathcal{H})$, and the new objects. If multiple basic sort labels are returned for an object, the lowest category in the sort hierarchy is used to add the new object to $\Pi(\mathcal{D}, \mathcal{H})$.

Next, the agent examines $\Pi(\mathcal{D}, \mathcal{H})$ to check whether the action verb (e.g., *grab*) from the LLM's output is already defined. If an action that semantically matches this action verb exists in $\Pi(\mathcal{D}, \mathcal{H})$, the agent retrieves it; this action may exhibit the same behavior as the one in the cue but have a different name, e.g., *pick(A,O)* instead of *grab(A,O)*. The agent performs strictly controlled verb synthesis with WordNet (Miller, 1995) to retrieve synonyms for the action verb from the cue and checks the synonyms with the Σ of $\Pi(\mathcal{D}, \mathcal{H})$. If a match is found, the action verb from the LLM output is replaced by the existing action in Σ. Also, the sorts of the arguments of the action extracted from the human cue may differ from those of the action in $\Pi(\mathcal{D}, \mathcal{H})$, e.g., they may be subsorts or parent sorts. If the sorts of arguments of the action in $\Pi(\mathcal{D}, \mathcal{H})$ are parent sorts of those in the cue, the agent uses the existing action as is; if not, the agent lifts the sort of the existing action's arguments to the new sorts and updates $\Pi(\mathcal{D}, \mathcal{H})$. If the extracted action (or its equivalent) does not exist in $\Pi(\mathcal{D}, \mathcal{H})$, the agent adds it with the lowest (i.e., most specific) sort labels for arguments. This process ensures that we do not introduce the same action multiple times with different sorts. This procedure is also repeated for literals, and the actions and literals are used to convert the extracted axioms to the appropriate format, e.g., 'a causes l' converted to '*holds(l,I+1) :- occurs(a,I)*'), replacing the ground sorts with variables and ensuring consistency between head and body. The axiom is then added to $\Pi(\mathcal{D}, \mathcal{H})$ if it does not exist.

Learning from observations. The second strategy enables the ad hoc agent to refine its knowledge based on observations obtained during plan execution. It extends prior work in our group for acquiring new knowledge in the context of scene understanding tasks (Mota et al., 2021), to ad hoc teamwork by combining *decision tree induction* with knowledge-based reasoning.

1. The agent selects an action a_I from the newly learned set of actions A, and an initial state of the environment S_I. It simulates the execution of a_I in S_I in its current domain to collect information about the outcomes (e.g., end state, an inconsistent outcome).
2. An expected outcome's absence indicates the absence of an executability condition; any additional effects indicate missing causal law(s). If all observations match expectations for different actions and states, the current knowledge is considered to be comprehensive.

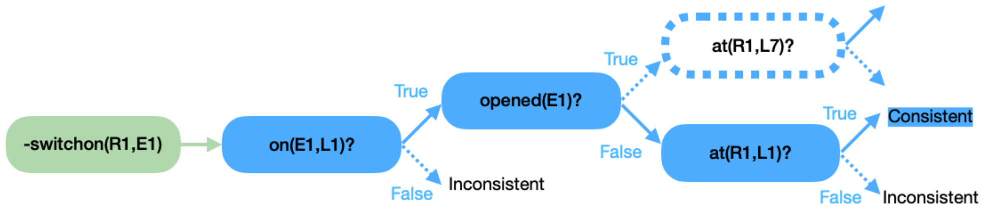


Figure 4: Part of the decision tree created to learn missing executability conditions.

3. The agent responds to an inconsistency by simulating the execution of a_I in different states, extracting from the answer set and initial state all those fluent literals that have an object constant that is also in a_I. These collected literals form part of the *training* examples.
4. In the training examples, the ground terms in literals are replaced by variables, and the dataset is reformatted with the fluent literals as features and the presence of absence of inconsistency as the class label. Each training example then records the presence or absence of a fluent literal, and the presence or absence of an inconsistency, as a binary value.
5. Separate decision trees are constructed for causal laws and executability conditions, with the action as the root node, and nodes are split using features that have not been used before and are likely to result in the highest reduction in entropy.
6. Candidate axioms are generated by traversing the learned trees from the root to the leaves using only those nodes that agree with their class label up to a threshold level (90%) and contain at least a minimum percentage (2%) of the dataset.
7. Only candidate axioms that have sufficient support among the training examples (90% in our experiments) are retained. Also the decision tree induction process is repeated multiple times over the training data to explore different subsets of data. Only axioms that are retained over multiple such repetitions are lifted to the more general form and added to $\Pi(\mathcal{D}, \mathcal{H})$.

Figure 4 shows part of a decision tree generated by this process, with the ad hoc agent learning the following two executability conditions:

$$-occurs(switchon(R, E), I) \leftarrow holds(on(E, L), I), not\ holds(at(R, L), I), \quad loc(L), agent(R), appliance(E). \tag{2a}$$

$$-occurs(switchon(R, E), I) \leftarrow holds(opened(E), I), agent(R), appliance(E). \tag{2b}$$

which imply that an agent cannot switch on an appliance if it is not in the same location or if the appliance's door is open. Although the learning strategy is described above in the contact of learning new objects, actions and axioms, it can be used to learn new literals (relations) too.

4. Experimental Setup and Results

We experimentally evaluated the following hypotheses regarding our architecture's capabilities:

H1 Reasoning with prior knowledge and the rapidly-learned behavior prediction models improves performance and promotes scalability;

H2 Using LLM-based anticipated tasks as joint goals improves performance compared with planning for one task at a time;

H3 Incrementally-updated prompts and validators improve task anticipation capability of the LLM and the team's performance;

H4 Using the LLM to directly output a sequence of low-level actions to complete assigned tasks results in poor performance;

H5 KAT enables the ad hoc agent to accurately learn unknown objects, actions, and axioms; and

H6 Reasoning with incrementally learned knowledge improves the performance of the team.

We evaluated these hypotheses in the *VirtualHome* domain. In each episode, the AI ad hoc agent(s) and a human agent collaborated to complete household tasks. We recorded the number of steps (plan length) and the task completion time as the *performance measures*. All prompts to the LLM were through OpenAI GPT4o mini API with default parameters (e.g., temperature = 1.0).

4.1 Experimental Setup

In our experiments, the human was modeled as a simulated entity whose action choices were based on an ASP program that considered the human's prior knowledge and runtime observations; the human did not reason with models predicting teammates' actions. Also, the human's ASP program encodes certain preferences, priorities, and capabilities that are not initially known to the ad hoc agents but may be captured over time in the models that the ad hoc agents learn in order to predict the behavior of the human. The sequence of tasks generated by the task generator were assigned to the agents over time; the agents were not aware of the complete sequence and received one task at a time. The human was assigned the same goal as the ad hoc agent(s). All agents received the same observations of the domain at each step, which they used to plan their respective actions. There was no direct communication between them.

When an ad hoc agent equipped with KAT received a task, it prompted the LLM (Section 3.3). The anticipated tasks were validated and mapped to ASP literals, with the next anticipated task and current task set as joint goals for this agent. During planning, the ad hoc agent also used the learned behavior prediction model to predict each teammate's actions for two future steps (Section 3.2). These predictive models were built using only 1000 examples of prior traces of actions and domain state. The agent initially assigns one learned model to each teammate but uses information from subsequent steps to incrementally revise models for each teammate based on their observed behavior. Predicted actions from the models are then mapped to exogenous actions that are added to the ASP program along with initial state information and refined sorts. The ASP program of the ad hoc agent included additional axioms for reasoning with the predicted actions of each teammate. As a result, the ad hoc agent's plan anticipated preconditions of some intermediate steps to be satisfied by a teammate's actions, even though the teammate did not always execute that action. The ad hoc agent hence had to respond to unexpected action outcomes and domain states.

To evaluate **H1** and **H2**, in **Exp1**, we randomly selected 100 task routines sampled from predefined sequences and measured the ability of a team (human and an ad hoc agent) to complete these tasks. Performance measures were the number of steps and time taken. We used three baselines:

- **Base1**: LLM for anticipating future tasks, but no behavior models to predict human's actions.

- **Base2**: no LLM to anticipate future tasks, but behavior models to predict the human's actions.
- **Base3**: did not use LLM for task anticipation or behavior models to predict human's actions.

In the absence of a framework for AHT that supported all capabilities of KAT, we chose these baselines to conduct ablation studies that evaluated the contribution of each key component of KAT. Since the actual time taken and the number of actions required to complete tasks can vary substantially based on the task, the average of these values over the individual trials may not be meaningful. We instead *ran paired trials and computed performance measure values for the baselines as a fraction of these values for KAT in each trial*. We then reported the average of these ratios.

For evaluating scalability in **H1**, we increased the team size by introducing additional ad hoc agents, with three agents (one human, two ad hoc agents) and four agents (one human, three ad hoc agents) collaborating to complete tasks (as in **Exp1**). These different configurations would normally make collaboration increasingly challenging, e.g., with just two agents the domain has $\approx 10^{25}$ possible states, and this number increases exponentially with the number of AI agents. We then measured the number of steps and time taken by the agent teams to complete the tasks.

To evaluate **H3**, in **Exp2**, we computed the precision and recall of the tasks anticipated by the LLM, before and after applying the validator, over the 100 task routines. We also computed the precision and recall of a simple statistical model that replaced the LLM and anticipated the agent's next task based on past experience. Further, in **Exp3**, we randomly selected 20 task routines and recorded the LLM's performance with and without prompting methods, using four baselines: **Base4**: no prompting strategy or validator; **Base5**: few-shot prompting but no validator; **Base6**: CoT prompting but no validator; and **Base7**: validator but no prompt-engineering.

For evaluating **H4**, we conducted experiment **Exp4**, in which we created an architecture that used the LLM to directly output sequences of actions for specific tasks (**Base8**). Specifically, our prompt included details of actions available in our example scenario in *VirtualHome*, their intended purpose (from ASP program, e.g., move(agent, location): move the agent to an adjacent location). We also supplied the LLM some **Action Feasibility Rules**:

- *Movement Limitation* (critical): must only move to adjacent locations defined by the next_to relationships. Always check adjacency before predicting a move.
- *Object Location*: must be in the same location as an object to act on it (e.g., grab, put).
- *Carrying Limit*: cannot hold more than two objects. When holding two objects, actions like open, close, switch-on, or switch-off require you to put at least one object down first.
- *Appliance Safety*: for safety, you should not open appliance doors when they are switched on.
- *Avoid Conflict*: if a human is holding an object, they will handle all actions with the object. Do not attempt to grab or interact with this object. Instead, focus on other parts of the goal.

We included information about adjacent places in the domain emphasizing the fact that the agent can only move between the defined adjacent places. The LLM also had access to the current world state, including the location of the agents, objects, and appliances, each appliance's state, and information about the objects held by the agents. The problem specification also described the task to be performed; the immediate previous actions of the human and the ad hoc agent; any specific information to be considered on any given day (e.g., human working from home). Additionally, the prompt included a detailed example of selecting an action, and asked the LLM to generate an action sequence for achieving the assigned goal and specify next action to execute.

The LLM's action choice was assigned as the ad hoc agent' action. As a recovery mechanism, we corrected errors in the LLM output up to three times per trial. For example, if the LLM's action involves grabbing an object without moving to the appropriate location, we provided feedback explaining why this choice was incorrect and allowed the LLM to predict another action for that step. We measured the performance of the agent team to complete the previously selected 100 task routines. The performance measures were the number of steps and time taken to complete the tasks.

In **Exp5**, we introduced another ad hoc agent with an incomplete knowledge base to the two agent team (ad hoc agent and human). The new agent's ASP program included only a subset of the objects (17/31), actions (4/7), and axioms (6/9 causal laws, 16/26 executability conditions). This corresponded to the absence of around $40 - 45\%$ knowledge. We made sure that this agent had enough initial knowledge to perform some basic activities, while also withholding key knowledge to create gaps that limited the agent's ability to complete tasks. During task execution, the human agent periodically describe actions of the knowledgeable ad hoc agent to the ad hoc agent with missing knowledge. This agent then used the procedure described in Section 3.4 to process these descriptions into ASP sorts, actions and axioms with the help of an LLM and added the validated information to its knowledge base. At the end of each episode, it also used decision tree induction to learn new axioms. We then evaluated the ad hoc agent's ability to learn missing knowledge across 10 episodes, with each episode randomly selecting from five different task sequences and each sequence consisting of four tasks. Similar to previous experiments, task sequences were generated by the task generator and provided to the agent one at a time. However, we intentionally omitted the future task anticipation algorithm to prevent its influence on knowledge acquisition capabilities. We recorded the number of objects, actions, and axioms learned in each trial, and the precision and recall of learning these axioms compared with a complete ASP program (ground truth).

In **Exp6**, we extended each episode from **Exp5** to include three consecutive runs. The first run in an episode had the same initial knowledge as in **Exp5**, but the subsequent two runs built on the knowledge for a potentially different sequence of tasks. This process was repeated for the 10 episodes; we recorded the objects, actions, and axioms learned after each run and episode, and computed precision and recall as the performance measures.

To evaluate **H6** in **Exp7**, we ran 20 trials with and without the learned axioms, recording the number of steps (plan length) and the time taken to complete the assigned task(s).

4.2 Experimental Results

Table 2 summarizes the results of **Exp1**. When the ad hoc agent reasoned with anticipated tasks and predicted actions, it provided the best performance with lowest number of action steps and least amount of time taken to complete task routines. The accuracy of the human behavior prediction models learned by the ad hoc agent was 85%, i.e., it makes errors, but it supports rapid learning and revision. Also, reasoning with prior knowledge and the output of these predictive models significantly improves performance. While other algorithms may achieve higher predictive accuracy, we chose FF tree models because they are simple, easy to understand, and can be revised rapidly. When the agent used task anticipation but not the behavior prediction models (**Base1**), the number of steps

Table 2: Average number of steps and time taken to complete task routines; values for baselines computed as a fraction of these values for KAT in each trial; for comparison, the average absolute values are 26.8 steps and 361 seconds for KAT.

Architecture	*Steps*	*Time(s)*
KAT (anticipate tasks, predict actions)	1.0	1.0
Base1 (anticipate tasks)	1.1	1.1
Base2 (predict actions)	1.3	1.2
Base3	1.4	1.4
Base8 (LLM predict low-level actions)	1.5	1.5

Table 3: Average number of steps and time taken by Team1, Team2 and Team3 to complete the task routines, with performance measure values for Teams 2-3 computed as a fraction of the values for Team 1 in each trial.

Team	*Steps*	*Time(s)*
Team1	1.0	1.0
Team2	0.8	0.9
Team3	0.7	0.8

and time taken increased; not considering the teammates' actions may lead the agent to waste time in executing redundant actions. These results emphasize the importance of the behavior prediction models, supporting hypothesis **H1**.

When the ad hoc agent used the behavior prediction models but did not anticipate future tasks (**Base2**), performance worsened, with a further increase in the number of steps and the time taken to complete tasks. Planning jointly for the current and anticipated tasks saved time and effort. For example, when the agent visited the bedroom to retrieve a board game for the guests, it also picked up bottles of wine from the cellar on the way instead of making two separate trips. These results support **H2**. Also, when the ad hoc agent did not use task anticipation or the behavior prediction models (**Base3**), the performance worsened further. These results support **H1** and **H2**. Finally, using the LLM to directly compute a sequence of low-level actions (**Base8**) resulted in the worst observed performance. All results were statistically significant with $p < 0.0001$. These results support **H4**.

Next, Table 3 summarizes the performance of **Team1** (human, one ad hoc agent), **Team2** (human, two ad hoc agents) and **Team3** (human, three ad hoc agents) in completing the same set of 100 task routines. As the number of ad hoc agents increases, task completion becomes more efficient: Team2 outperformed Team1 by requiring fewer steps and less time to complete the tasks, while Team3 showed further improvements over Team2. These results emphasize the importance of efficient collaboration and demonstrate the scalability of the architecture to multiple agents, supporting **H1**. Once again all the results were statistically significant with $p < 0.0001$. The observed performance was primarily due to design choices in KAT to enable each ad hoc agent to reason independently and efficiently using domain knowledge and learned models.

Table 4 shows the results from **Exp2**, where we computed the precision and recall values of the tasks anticipated by the LLM before and after applying the validator. We observed marked improvement in precision after applying the validator to the LLM's output. The errors in the LLM's outputs were substantially reduced by the validator. The recall values do not change substantially as the validator did not introduce new tasks; it only reordered the tasks that are out of order and removed irrelevant tasks. On the other hand, the simple statistical model that anticipated future tasks based only on past experience resulted in precision and recall of 0.75 and 0.76 respectively. These results indicate that using just the historical data is insufficient for task anticipation, although performance is comparable with the raw output obtained from the LLM; future work will explore

Table 4: Precision and recall values of the anticipated tasks with and without the LLM and the validator. A simple statistical model is not good enough and the validator plays an important role.

Architecture	*Precision*	*Recall*
Simple statistical model	0.75	0.76
LLM without validator	0.78	0.76
LLM with validator	0.99	0.78

Table 5: Average number of steps and time taken by the team (human, ad hoc agent) to complete tasks with prompting strategies and/or validator; values of performance measures for baselines computed as a fraction of values for KAT in each trial, with the average absolute values of 27.5 steps and 372.7 seconds for KAT.

Architecture	*Steps*	*Time(s)*
KAT (all prompting, validator)	1.00	1.00
Base4 (no prompting, no validator)	1.21	1.15
Base5 (few-shot, no validator)	1.17	1.18
Base6 (CoT, no validator)	1.16	1.16
Base7 (no prompting, validator)	1.05	1.04

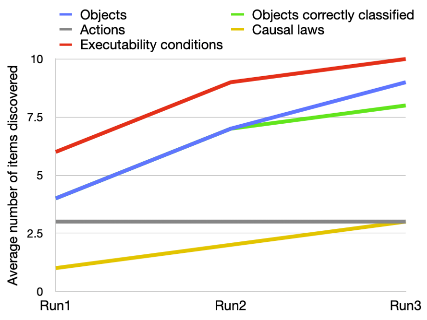


Figure 5: Average number of objects, actions, axioms learned in three consecutive runs, averaged over 10 episodes.

the use of the simpler model in more complex domains. Using the validator to correct the LLM's output based on prior knowledge and experiences leads to better results, supporting **H3**.

Table 5 shows the results from **Exp3**, which explored the use of different prompting strategies and the validator with the LLM to anticipate future tasks (Section 3.3). We again observed a significant improvement in performance, e.g., a lower number of steps and time to complete tasks with the external validator and a combination of prompting methods. In particular, using the validator to adapt the LLM's output to the domain had a significant impact on performance, further supporting **H3**. These results were statistically significant for Base4, Base5 and Base6 with $p < 0.001$. For Base7 the results were mixed; the reduction in steps was statistically significant ($p < 0.05$), while the reduction in time was not ($p = 0.09$). This outcome is consistent with expectations since Base7 used the validator; it emphasizes the importance of the validator and the need for further exploration of more complex scenarios to determine the importance of the prompting strategies.

Results of **Exp5** are summarized in Table 6. We observed high precision and recall for learning the missing axioms. Figure 5 summarized the results of **Exp6** with three consecutive runs in each of 10 episodes. By the end of the first run, the agent successfully learned 4-5 of 14 missing objects, all three missing actions, 1-2 of three missing causal laws, and 6–7 of 10 missing executability conditions. After three runs, the values increased to 8–9 objects, 2–3 causal laws, and 9–10 executability conditions. The steady increase in number of objects, actions, and axioms, along with the high precision and recall, indicated the agent's ability to reliably learn new knowledge, supporting **H5**. Although the LLM occasionally provided incorrect ASP formats, these errors occurred rarely;

Table 6: Average precision and recall of learned axioms in 30 runs over 10 episodes; previously unknown axioms were learned accurately.

Axiom type	*Precision*	*Recall*
Causal laws	0.96	1.0
Executability conditions	1.0	1.0

Table 7: Average number of steps and time taken by team to complete task sequences represented as a fraction of these values for baseline.

Architecture	*Steps*	*Time(s)*
With learned axioms	0.76	0.84
Without learned axioms	1.00	1.00

most of the time the LLM output was correct as its use was limited to straightforward tasks and the mechanisms described in Section 3.4 were sufficient to account for these errors.

Table 7 summarizes the results of **Exp7**, which evaluated the impact of the learned knowledge on the team's ability to complete tasks. We ran paired trials with and without the learned axioms and computed performance measure values for the former as a fraction of the values for the latter. Reasoning with the learned objects, actions, and axioms substantially improved performance. In the absence of the learned knowledge, at least one ad hoc agent often could not compute valid plans to complete the tasks, and could not contribute to the team's performance. The team was essentially operating with one fewer member in such cases, with the other two members executing additional actions. The significant low number of steps and time ($p < 0.0001$) emphasize the importance of learning previously unknown knowledge, thus supporting **H6**.

4.3 Execution Traces

Knowledge Acquisition: We provide some execution traces as a qualitative evaluation of **H5**. Consider the situation in which the ad hoc agent was unaware of the action *grab(agent, object)* that refers to an object being picked up. When the human provided the cue, "Agent2 grabbed the apple", the agent used the process described in Section 3.4 to obtain the following output from the LLM:

$$grab(agent2, apple)\ causes\ in\_hand(agent2, apple). \quad (3)$$

From this LLM output, the agent first extracted *grab(agent2, apple)*. Since *apple* is not already defined in its current knowledge base (ASP program), the agent sent this to the LLM (message 2 in Figure 2) to receive the output *apple: food*, which states that *apple* belongs to the sort *food*. The ad hoc agent then generalized this action to *grab(agent, food)* based on the LLM output, and added the action to its knowledge. Further, the agent also learned the causal law:

$$holds(in\_hand(R, F), I + 1) \leftarrow occurs(grab(R, F), I), agent(R), food(F). \quad (4)$$

In subsequent runs, the ad hoc agent applied the grab action to food items, but remained unaware that this action was applicable to other objects such as a book or a cellphone. As a result, it did not contribute to tasks such as preparing the workstation. However, since the agents executed tasks as a team, the human description of the execution of *grab(agent,book)* by another agent enabled the ad hoc agent to generalize the action to *grab(agent, graspable)* and update the causal law:

$$holds(in\_hand(R, G), I + 1) \leftarrow occurs(grab(R, G), I), agent(R), graspable(G). \quad (5)$$

The agent could have also learned the more general version of this axiom (Statement 5) over time.

Once the agent learned above causal laws, it was able to used them to compute plans and complete tasks even though it was unaware of all the executability conditions for this action. Later the human provided the cues “Agent2 cannot grab the plate since plate is on the kitchen_table and Agent2 is not at kitchen_table" and “Agent2 moved to kitchen_table". The agent used the LLM to translate these cues to the following executability condition and causal law:

$$-\,grab(agent2, plate)\ \textbf{if}\ on(plate, kitchen\_table),\ \textit{not}\ at(agent2, kitchen\_table). \tag{6a}$$

$$move(agent2, kitchen\_table)\ \textbf{causes}\ \ at(agent2, kitchen\_table). \tag{6b}$$

The ad hoc agent used this output to learn the following axioms:

$$\begin{aligned} -occurs(grab(R,G),I) \leftarrow & holds(on(G,L),I), notholds(at(R,L),I),\ agent(R), \\ & location(L),\ graspable(G). \end{aligned} \tag{7a}$$

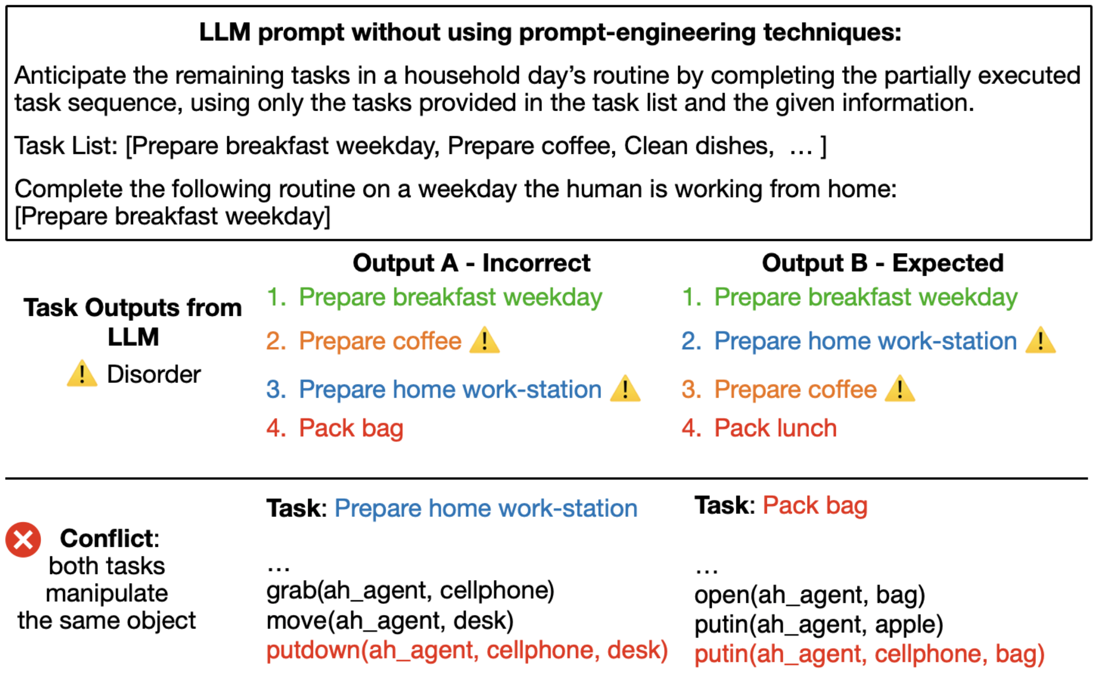
$$holds(at(R,L),I+1) \leftarrow occurs(move(R,L),I), agent(R), location(L). \tag{7b}$$

Since Statement 7(b) was already in its knowledge base, the ad hoc agent only added Statement 7(a) to its knowledge base. These results support **H5**, i.e., that the agent is able to incrementally learn and generalize the learned knowledge to improve task completion performance.

Task Anticipation: We also provide some execution traces as a qualitative evaluation of **H2**, **H3** and **H4**. Figure 6 shows an execution example where the ad hoc agent used the LLM to anticipate future tasks with and without the prompting strategies and validator. The example was set on a weekday where the human was working from home and no guests were expected. The correct task routine in this context was: *Prepare breakfast, Prepare home work-station, Prepare coffee, Prepare lunch*.

When the ad hoc agent queried the LLM without the prompt engineering techniques or validator (Section 3.3), the anticipated task list was different from the expected output. The prompt to the LLM without using the prompt engineering strategies is shown in Figure 6. The LLM output was [*Prepare breakfast, Prepare coffee, Prepare home work-station, Pack bag*]. This output failed to align with the human preferences and priorities because: (a) making coffee was assigned higher priority than setting up the workstation, which would delay when the human started work and the coffee would not be hot when needed; and (b) it was not necessary to pack the bag since the human was working from home. On the other hand, when the ad hoc agent used the prompt engineering strategies and the validation strategy, the prompt to the LLM was automatically generated while incorporating context, as described in Section 3.3. The LLM's output was [*Prepare breakfast, Prepare home work-station, Prepare coffee, Prepare lunch*]. This matched the expected routine. i.e., making breakfast and setting up the workstation were considered high priority tasks, and irrelevant tasks such as *pack bag* were filtered out by the validator. These results demonstrate the importance of using a combination of prompting techniques and the external validator, supporting **H3**.

We observed similar situations when extending the setup to three agents, one human and two ad hoc agents, collaborating on a weekend when guests were expected; the correct task routine was: *Prepare breakfast, Prepare table for guests, Prepare lunch, Clean dishes*. When the first ad hoc agent queried the LLM with the prompting strategies but without the validator, the anticipated task list by the LLM was *Prepare breakfast, Prepare table for guests, Prepare lunch, Serve snacks*.


LLM prompt without using prompt-engineering techniques:

Anticipate the remaining tasks in a household day's routine by completing the partially executed task sequence, using only the tasks provided in the task list and the given information.

Task List: [Prepare breakfast weekday, Prepare coffee, Clean dishes, …]

Complete the following routine on a weekday the human is working from home:
[Prepare breakfast weekday]

Task Outputs from LLM ⚠ Disorder	Output A - Incorrect	Output B - Expected
	1. Prepare breakfast weekday	1. Prepare breakfast weekday
	2. Prepare coffee ⚠	2. Prepare home work-station ⚠
	3. Prepare home work-station ⚠	3. Prepare coffee ⚠
	4. Pack bag	4. Pack lunch

✗ **Conflict:** both tasks manipulate the same object

Task: Prepare home work-station

```
…
grab(ah_agent, cellphone)
move(ah_agent, desk)
putdown(ah_agent, cellphone, desk)
```

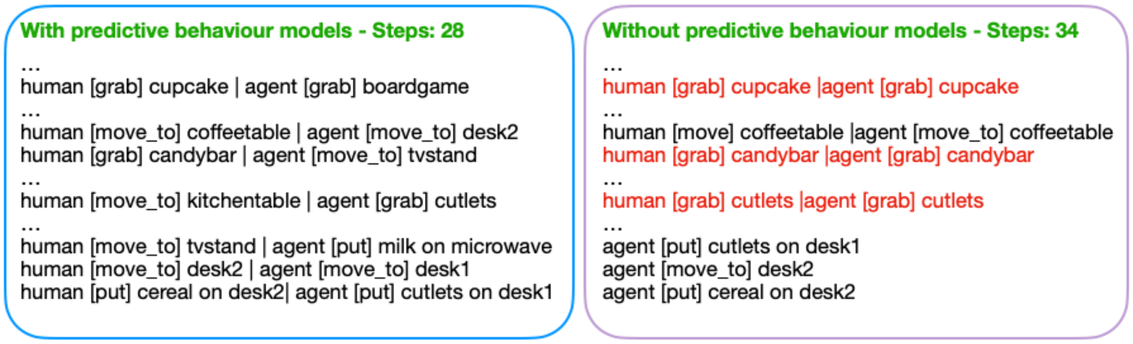
Task: Pack bag

```
…
open(ah_agent, bag)
putin(ah_agent, apple)
putin(ah_agent, cellphone, bag)
```

Figure 6: Execution example: using LLM without prompting strategies or external validator causes conflicts during execution, having a negative impact on performance.

For the second ad hoc agent the LLM output was *Prepare breakfast, Prepare table for guests, Prepare lunch, Serve snacks, Clean dishes*. When the validator was used, the outputs to both agents were refined by incorporating context. Since the human usually did not require snacks after lunch, the *Serve snacks* task was removed. The refined task list was *Prepare breakfast, Prepare table for guests, Prepare lunch* for first ad hoc agent, and *Prepare breakfast, Prepare table for guests, Prepare lunch, Clean dishes* for the second ad hoc agent; since *Clean dishes* was a defined task, it was retained by the validator. This example further demonstrates the importance of using the validator, and supports hypothesis **H3**.

Figure 7 compares two plans executed by a team comprising a human and an ad hoc agent for completing a different set of tasks: *[Prepare breakfast, Prepare activities, Serve snacks, Clean kitchen]*, with and without the behavior prediction models (Section 3.2). In the first plan, when the ad hoc agent used the behavior prediction model to predict the future actions of the human, the team successfully completed all tasks for the given day in just 28 steps. On the other hand, when the ad hoc agent did not use behavior prediction models, it often selected the same actions as the human for any particular task, leading to unnecessary delays in completing the tasks. For example, in the second plan the agent frequently selected the same action as the human—simultaneously picking up the cupcake, candy bar, and cutlets, introducing redundant behavior and prolonging task execution. As a result, the overall plan was extended to 34 steps. These results demonstrate that using the behavior prediction models enables the ad hoc agent to coordinate efficiently by avoiding action conflicts with the human. This further supports **H2**.



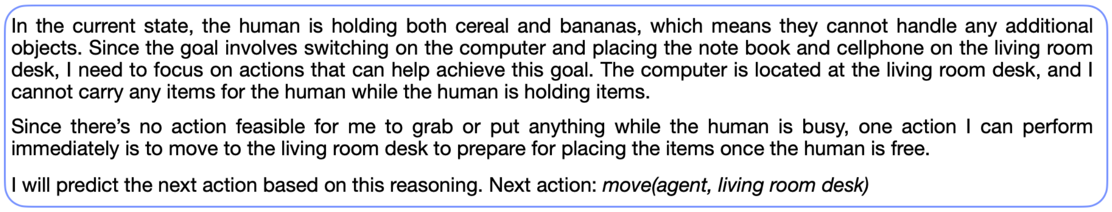
With predictive behaviour models - Steps: 28

```
...
human [grab] cupcake | agent [grab] boardgame
...
human [move_to] coffeetable | agent [move_to] desk2
human [grab] candybar | agent [move_to] tvstand
...
human [move_to] kitchentable | agent [grab] cutlets
...
human [move_to] tvstand | agent [put] milk on microwave
human [move_to] desk2 | agent [move_to] desk1
human [put] cereal on desk2| agent [put] cutlets on desk1
```

Without predictive behaviour models - Steps: 34

```
...
human [grab] cupcake |agent [grab] cupcake
...
human [move] coffeetable |agent [move_to] coffeetable
human [grab] candybar |agent [grab] candybar
...
human [grab] cutlets |agent [grab] cutlets
...
agent [put] cutlets on desk1
agent [move_to] desk2
agent [put] cereal on desk2
```

Figure 7: Execution trace for task routine: *[Prepare breakfast, Prepare activities, Serve snacks, Clean kitchen].* When an ad hoc agent is not allowed to predict and reason about the human's actions, it may choose to execute same action(s) as the human, leading to longer plans.



In the current state, the human is holding both cereal and bananas, which means they cannot handle any additional objects. Since the goal involves switching on the computer and placing the note book and cellphone on the living room desk, I need to focus on actions that can help achieve this goal. The computer is located at the living room desk, and I cannot carry any items for the human while the human is holding items.

Since there's no action feasible for me to grab or put anything while the human is busy, one action I can perform immediately is to move to the living room desk to prepare for placing the items once the human is free.

I will predict the next action based on this reasoning. Next action: *move(agent, living room desk)*

Figure 8: Execution example: using LLM to generate low-level actions results in poor performance.

When the ad hoc agent used the LLM to directly output action sequences for specific tasks (**Base8**), prompts were constructed as described in Section 4.1. Figure 8 shows part of the LLM output during the task routine: [*Prepare breakfast, Prepare home work-station, Prepare coffee, Prepare lunch*]. The action *move(agent, living room desk)* violated the 'Movement Limitation (Critical)' rule in 'Action Feasibility Rules' (Section 4.1), which constrained the agent to only move to locations adjacent to its current location. This example demonstrates that the LLM may not respect constraints even when they are provided as input, and highlights that an LLM is not designed for computing plans for non-trivial tasks; using an LLM to directly output a sequence of low-level actions to complete tasks can lead to poor performance, which supports hypothesis **H4**.

Source code and additional results for our core architecture are in our **open-source repository** (Dodampegama & Sridharan, 2025b); for code and results related to knowledge acquisition, please see (Dodampegama & Sridharan, 2025a).

5. Conclusions

This paper described KAT, an architecture for Ad Hoc Teamwork (AHT) that enables an AI agent to collaborate with other agents (human, AI) in complex domains without prior coordination. KAT integrates the principles of refinement, ecological rationality, and interactive learning, enabling the agent to: automatically identify and reason with relevant information; effectively leverage the generic knowledge encoded in an LLM for high-level task anticipation; rapidly learn models predicting the action choices of its teammates; perform non-monotonic logical reasoning with prior knowledge and the behavior prediction models to jointly plan and execute actions to achieve the current and anticipated tasks; leverage a LLM to translate natural language descriptions of action outcomes into formal ASP representations of previously unknown objects, actions, and axioms; and use decision tree induction to incrementally learn and revise axioms based on observations. Based on experiments in a realistic, physics-based simulation environment, we demonstrated the architecture's capabilities compared with various baselines, highlighting the significance of each component of our architecture, and the promising ability to scale to additional agents. Future work will extend this approach to incorporate a human controlled avatar in the VirtualHome simulation environment, and to physical robots collaborating with humans in AHT settings.

Acknowledgements

This work was supported in part by the U.S. Office of Naval Research Award N00014-20-1-2390. All conclusions described in this paper are those of the authors alone.

References

Balai, E., Gelfond, M., & Zhang, Y. (2013). Towards Answer Set Programming with Sorts. *Conference on Logic Programming and Nonmonotonic Reasoning*.

Barrett, S., Rosenfeld, A., Kraus, S., & Stone, P. (2017). Making friends on the fly: Cooperating with new teammates. *Artificial Intelligence*, *242*, 132–171.

Barrett, S., Stone, P., Kraus, S., & Rosenfeld, A. (2013). Teamwork with limited knowledge of teammates. *AAAI Conference on Artificial Intelligence*.

Bowling, M., & McCracken, P. (2005). Coordination and adaptation in impromptu teams. *National Conference on Artificial Intelligence* (p. 53–58).

Chen, S., Andrejczuk, E., Cao, Z., & Zhang, J. (2020). AATEAM: Achieving the ad hoc teamwork by employing the attention mechanism. *AAAI*.

Dodampegama, H., & Sridharan, M. (2023). Knowledge-based Reasoning and Learning under Partial Observability in Ad Hoc Teamwork. *Theory and Practice of Logic Programming*, *23*, 696–714.

Dodampegama, H., & Sridharan, M. (2025a). `https://github.com/hharithaki/Knowledge-Acquisition`.

Dodampegama, H., & Sridharan, M. (2025b). `https://github.com/hharithaki/Task-Anticipation`.

Fang, Q., Zeng, J., Xu, H., Hu, Y., & Yin, Q. (2024). Learning ad hoc cooperation policies from limited priors via meta-reinforcement learning. *Applied Sciences*, *14*.

Gelfond, M., & Inclezan, D. (2013). Some Properties of System Descriptions of AL_d. *Applied Non-Classical Logics, Special Issue on Equilibrium Logic and ASP*, *23*.

Gelfond, M., & Kahl, Y. (2014). *Knowledge Representation, Reasoning and the Design of Intelligent Agents*. Cambridge University Press.

Gigerenzer, G. (2016). *Towards a Rational Theory of Heuristics*, (pp. 34–59). London: Palgrave Macmillan UK.

Gigerenzer, G. (2020). What is Bounded Rationality? In *Routledge Handbook of Bounded Rationality*. Routledge.

Kambhampati, S., Valmeekam, K., Guan, L., Verma, M., Stechly, K., Bhambri, S., Saldyt, L., & Murthy, A. (2024). Llms can't plan, but can help planning in llm-modulo frameworks.

Katsikopoulos, K., Simsek, O., Buckmann, M., & Gigerenzer, G. (2021). *Classification in the Wild: The Science and Art of Transparent Decision Making*. MIT Press.

Liu, X., Li, P., Yang, W., Guo, D., & Liu, H. (2024). Leveraging Large Language Model for Heterogeneous Ad Hoc Teamwork Collaboration. *Robotics: Science and Systems Conference*. Delft, The Netherlands.

Miller, G. A. (1995). Wordnet: a lexical database for english. *Commun. ACM*, *38*, 39–41.

Mirsky, R., Carlucho, I., Rahman, A., Fosong, E., Macke, W., Sridharan, M., Stone, P., & Albrecht, S. (2022). A Survey of Ad Hoc Teamwork: Definitions, Methods, and Open Problems. *European Conference on Multiagent Systems*.

Mota, T., Sridharan, M., & Leonardis, A. (2021). Integrated Commonsense Reasoning and Deep Learning for Transparent Decision Making in Robotics. *Springer Nature CS*, *2*.

OpenAI, et al. (2024). GPT-4 Technical Report.

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

Rahman, M. A., Hopner, N., Christianos, F., & Albrecht, S. V. (2021). Towards open ad hoc teamwork using graph-based policy learning. *International Conference on Machine Learning* (pp. 8776–8786).

Simon, H. A. (1956). Rational Choice and the Structure of the Environment. *Psychological Review*, *63*, 129–138.

Sridharan, M., Gelfond, M., Zhang, S., & Wyatt, J. (2019). REBA: A Refinement-Based Architecture for Knowledge Representation and Reasoning in Robotics. *Journal of Artificial Intelligence Research*, *65*, 87–180.

Stone, P., Kaminka, G., Kraus, S., & Rosenschein, J. (2010). Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination. *AAAI Conference on AI* (pp. 1504–1509).

Xu, P., Zhang, Y., Hao, L., & Yan, Q. (2025). DETEAMSK: A Model-Based Reinforcement Learning Approach to Intelligent Top-Level Planning and Decisions for Multi-Drone Ad Hoc Teamwork by Decoupling the Identification of Teammate and Task. *Aerospace*, *12*.

Zintgraf, L., Devlin, S., Ciosek, K., Whiteson, S., & Hofmann, K. (2021). Deep interactive bayesian reinforcement learning via meta-learning. *International Conference on Autonomous Agents and Multiagent Systems*.